Université de Montréal

Approches bio-informatiques appliquées aux technologies émergentes en génomique

par
Louis-Philippe Lemieux Perreault

Département de biochimie
Faculté de médecine

Thèse présentée à la Faculté des études supérieures
en vue d'obtention du grade de docteur
en bio-informatique

Hiver 2014

Université de Montréal
Faculté des études supérieures

Cette thèse intitulée :

Approches bio-informatiques appliquées

aux technologies émergentes en génomique

présenté par :

Louis-Philippe Lemieux Perreault

a été évaluée par un jury composé des personnes suivantes :

Damian Labuda, Ph. D.
président-rapporteur

Marie-Pierre Dubé, Ph. D.
directeur de recherche

Philip Awadalla, Ph. D.
codirecteur

Brian Wilhelm, Ph. D.
membre du jury

Guillaume Bourque, Ph. D.
examinateur externe

John Rioux, Ph. D.
représentant du doyen de la FES

Abstract

Genetic studies, such as linkage and association studies, have contributed greatly to a better understanding of the etiology of several diseases. Nonetheless, despite the tens of thousands of genetic studies performed to date, a large part of the heritability of diseases and traits remains unexplained. The last decade experienced unprecedented progress in genomics. For example, the use of microarrays for high-density comparative genomic hybridization has demonstrated the existence of large-scale copy number variations and polymorphisms. These are now detectable using DNA microarray or high-throughput sequencing. In addition, high-throughput sequencing has shown that the majority of variations in the exome are rare or unique to the individual. This has led to the design of a new type of DNA microarray that is enriched for rare variants that can be quickly and inexpensively genotyped in high throughput capacity.

In this context, the general objective of this thesis is the development of methodological approaches and bioinformatics tools for the detection at the highest quality standards of copy number polymorphisms and rare single nucleotide variations. It is expected that by doing so, more of the missing heritability of complex traits can then be accounted for, contributing to the advancement of knowledge of the etiology of diseases.

We have developed an algorithm for the partition of copy number polymorphisms, making it feasible to use these structural changes in genetic linkage studies with family data. We have also conducted an extensive study in collaboration with the *Wellcome Trust Centre for Human Genetics* of the *University of Oxford* to characterize rare copy number definition metrics and their impact on study results with unrelated individuals. We have conducted a thorough comparison of the performance of genotyping algorithms when used with a new DNA microarray composed of a majority of very rare genetic variants. Finally, we have developed a bioinformatics tool for the fast and efficient

processing of genetic data to increase quality, reproducibility of results and to reduce spurious associations.

Key words: Bioinformatics, Single nucleotide variants and polymorphisms, Copy number variations and polymorphisms, DNA microchip, Genetic data quality control

Résumé

Les études génétiques, telles que les études de liaison ou d'association, ont permis d'acquérir une plus grande connaissance sur l'étiologie de plusieurs maladies affectant les populations humaines. Même si une dizaine de milliers d'études génétiques ont été réalisées sur des centaines de maladies ou autres traits, une grande partie de leur héritabilité reste inexpliquée. Depuis une dizaine d'années, plusieurs percées dans le domaine de la génomique ont été réalisées. Par exemple, l'utilisation des micropuces d'hybridation génomique comparative à haute densité a permis de démontrer l'existence à grande échelle des variations et des polymorphismes en nombre de copies. Ces derniers sont maintenant détectables à l'aide de micropuce d'ADN ou du séquençage à haut débit. De plus, des études récentes utilisant le séquençage à haut débit ont permis de démontrer que la majorité des variations présentes dans l'exome d'un individu étaient rares ou même propres à cet individu. Ceci a permis la conception d'une nouvelle micropuce d'ADN permettant de déterminer rapidement et à faible coût le génotype de plusieurs milliers de variations rares pour un grand ensemble d'individus à la fois.

Dans ce contexte, l'objectif général de cette thèse vise le développement de nouvelles méthodologies et de nouveaux outils bio-informatiques de haute performance permettant la détection, à de hauts critères de qualité, des variations en nombre de copies et des variations nucléotidiques rares dans le cadre d'études génétiques. Ces avancées permettront, à long terme, d'expliquer une plus grande partie de l'héritabilité manquante des traits complexes, poussant ainsi l'avancement des connaissances sur l'étiologie de ces derniers.

Un algorithme permettant le partitionnement des polymorphismes en nombre de copies a donc été conçu, rendant possible l'utilisation de ces variations structurales dans le cadre d'étude de liaison génétique sur données familiales. Ensuite, une étude exploratoire a permis de caractériser les différents problèmes associés aux études génétiques utilisant des variations en nombre de copies

rares sur des individus non reliés. Cette étude a été réalisée avec la collaboration du *Wellcome Trust Centre for Human Genetics* de l'*University of Oxford*. Par la suite, une comparaison de la performance des algorithmes de génotypage lors de leur utilisation avec une nouvelle micropuce d'ADN contenant une majorité de marqueurs rares a été réalisée. Finalement, un outil bio-informatique permettant de filtrer de façon efficace et rapide des données génétiques a été implémenté. Cet outil permet de générer des données de meilleure qualité, avec une meilleure reproductibilité des résultats, tout en diminuant les chances d'obtenir une fausse association.

Mots clés : Bio-Informatique, Variations et Polymorphismes nucléotidiques, Variations et Polymorphismes en nombre de copies, Micropuces d'ADN, Nettoyage de données génétiques

Table des matières

Liste des figures

Liste des tableaux

Liste des abréviations

ADN : **A**cide **D**ésoxyribo **N**ucléique

aCGH : puce d'hybridation génomique comparative (**a**rray-**C**omparative **G**enomic **H**ybridization)

BAC : chromosome bactérien artificiel (**B**acterial **A**rtificial **C**hromosome)

BAF : fréquence de l'allèle B (**B** **A**llele **F**requency)

BIR : réplication induite par cette coupure (**B**reak-**I**nduced **R**eplication)

CEU : résidents de l'Utah originaires de l'Europe du Nord et de l'Ouest (CEPH)

CHB : Chinois Han de Beijing, Chine

CNP : polymorphisme du nombre de copies (**C**opy **N**umber **P**olymorphism)

CNV : variation du nombre de copies (**C**opy **N**umber **V**ariation)

CRT : **C**yclic **R**eversible **T**ermination

DRMAA : **D**istributed **R**esource **M**anagement **A**pplication **A**PI

FISH : hybridation *in situ* en fluorescence (**F**luorescence ***I****n* ***S****itu* **H**ybridization)

GWAS : étude d'association pangénomique (**G**enome-**W**ide **A**ssociation **S**tudy)

HMM : modèle de Markov caché (**H**idden **M**arkov **M**odel)

IBD : identique par descente (**I**dentical **B**y **D**escent)

IBS : identique par état (**I**dentical **B**y **S**tate)

JPT : Japonais de Tokyo, Japon

kb : **k**ilo-**b**ase

LD : déséquilibre de liaison (**L**inkage **D**isequilibrium)

LQTS : syndrome du QT long (**L**ong **QT** **S**yndrome)

LRR : **L**og **R** **R**atio

LVOTO : obstruction dynamique intraventriculaire gauche (**L**eft **V**entricular **O**utflow **T**ract **O**bstruction)

Mb : **m**ega-**b**ase

MMBIR : réplication induite par une coupure de l'ADN médiée par microhomologie (**M**icrohomology-**M**ediated **B**reak-**I**nduced **R**eplication)

MAF : fréquence de l'allèle mineur (**M**inor **A**llele **F**requency)

NAHR : recombinaison homologue non allélique (**N**on-**A**llelic **H**omologous **R**ecombination)

NHEJ : jonction d'extrémités non homologues (**N**on-**H**omologous **E**nd-**J**oining)

FoSTeS : blocage de la fourche de réplication et changement de matrice (**F**ork **S**talling and **Te**mplate **S**witching)

PCA : analyse par composante principale (**P**rincipal **C**omponent **A**nalysis)

Pb : **P**aires de **b**ases

QTL : locus à trait quantitatif (**Q**uantitative **T**rait **L**oci)

RFLP : polymorphisme de longueur des fragments de restriction (**R**estriction **F**ragment **L**ength **P**olymorphism)

ROMA : **R**epresentation **O**ligonucleotide **M**icroarray **A**nalysis

SBL : séquençage par ligation (**S**equencing **B**y **L**igation)

SNP : polymorphisme nucléotidique (**S**ingle **N**ucleotide **P**olymorphism)

SNV : variation nucléotidique (**S**ingle **N**ucleotide **V**ariation)

WGTP : **W**hole-**G**enome **T**iling **P**ath

WTCCC : **W**ellcome **T**rust **C**ase **C**ontrol **C**onsortium

YRI : Yoruba d'Ibadan, Nigeria

Dédicaces

« If I have seen further it is by standing on ye sholders of Giants. »

Isaac Newton

À ma copine, ma famille mes amis(ies) et mes collègues...

Remerciements

Pour commencer, je voudrais remercier ma directrice de recherche, Dre Marie-Pierre Dubé. Elle a su me guider tout au long de mon cheminement au doctorat et m'a permis de participer à des conférences internationales et de réaliser des collaborations des plus intéressantes. Par son soutien, ses conseils, son dynamisme et son expérience, elle a su accroître ma curiosité et mon intérêt au domaine de la génomique et de la médecine spécialisée. Ensuite, je voudrais remercier mon codirecteur, Dr Philip Awadalla, pour son soutien et ses conseils tout au long de mes études doctorales. Je voudrais aussi remercier Peter Donnelly ainsi que son équipe, dont Chris et Eleni, pour l'accueil qu'ils m'ont accordé.

Je voudrais ensuite remercier l'intégralité de l'équipe StatGen et du Centre de Pharmacogénomique, pour leur complicité et leur aide. Un merci spécial à Amina, Géraldine, Sylvie et Yassamin qui m'ont beaucoup aidé depuis le tout premier jour de mes études doctorales.

Un immense merci à mon amoureuse, Marie-Josée. Sa patience et son soutien au cours de mes études universitaires ont été des plus appréciés. Finalement, je voudrais remercier mes parents, ma soeur, ainsi que toute ma famille. Tous m'ont appuyé et ont toujours cru en moi et en mes compétences.

Merci !

Première partie

Introduction

1. Variations génétiques

Le génome humain est composé de séquences d'ADN d'environ trois milliards de nucléotides séparés en 23 paires de chromosomes. Selon les bases de données récentes, il contient un peu plus de 20 000 gènes encodant des protéines. Les régions codant pour des protéines ne représentent qu'une faible fraction du génome (soit environ 1,5 %), le reste étant composé de molécules d'ARN non codant, de séquences régulatrices, d'introns ou d'autres séquences dont la fonction est encore indéterminée [17]. Il a été estimé que toute paire de génomes humains ne différerait que par 0,1 % de ses nucléotides [18]. Ces différences sont constituées de variations génétiques ou de polymorphismes, généralement classés selon leur fréquence dans une population ; toute variation présente chez 1 % ou plus dans une population est nommée un polymorphisme. Il a été estimé qu'un individu typique possède, en moyenne, plus de 2 500 variations non synonymes (c.-à-d. une variation modifiant le produit d'un gène), de 20 à 40 variations identifiées comme dommageables et situées dans une région conservée du génome ainsi qu'environ 150 variations qui entraîneraient une diminution ou une perte totale de fonction [19]. La majorité de ces variations sont communes (plus de 5 %) ou de faible fréquence (entre 0,5 et 5 %). Si l'on ne considère que les variations rares (moins de 0,5 %), le fardeau diminue pour atteindre entre 130 et 400 variations non synonymes, entre 2 et 5 variations dommageables et entre 10 et 20 variations entraînant une perte de fonction chez un individu typique [19]. En analysant les données génétiques provenant de deux trios de populations européennes et africaines, il a été déterminé qu'un total de 49 et de 35 mutations *de novo* (ou personnelles) étaient présentes chez les enfants, respectivement. La majorité de ces variations se retrouvent dans les régions intergéniques (51,0 % et 48,6 %) ou introniques (38,8 % et 48,6 %) [20].

1.1. Historique

Depuis les années 1960, de nombreuses avancées scientifiques ont permis d'acquérir plus d'information concernant la variabilité du génome humain (Figure 1.1). Kan et Dozy furent les premiers, en 1978, à décrire une variation génétique impliquant uniquement un nucléotide (SNP) situé dans le site de reconnaissance de l'enzyme de restriction *HpaI*, en aval du gène de la β-globine [21]. Deux ans plus tard, en 1980, Wyman et White décrivirent un autre type de variation génétique : les polymorphismes de longueur des fragments de restriction (*restriction fragment length polymorphism* ou RFLP) [22]. En 1985, Jeffreys et coll. rapportèrent la découverte d'une classe de séquences d'ADN répétitives, composée de répétitions de deux à huit nucléotides organisées en tandem, qu'ils nommèrent minisatellites [23]. C'est en 1989 que plusieurs chercheurs décrivirent les microsatellites, une séquence d'ADN formée par une répétition continue de motifs composés de sept à cent nucléotides sur une longueur de 0,5 à 30 kb [24]. Toutes ces variations du génome humain furent utilisées dans les années 1990 afin de créer des cartes de liaison des chromosomes humains. Elles furent aussi utilisées afin de cartographier des mutations causales dans plus de 2 000 gènes [1]. Bien que l'existence des variations en nombre de copies (CNV) est connue depuis les années 1975 [25, 26], ce n'est que tout récemment, en 2004, que fut démontrée l'existence à grande échelle de ces variations structurales par l'utilisation de l'hybridation génomique comparative [27, 28]. Depuis, plusieurs études concernant les SNP et les CNV furent réalisées.

1.2. Polymorphismes et variations nucléotidiques

Les polymorphismes du génome humain les plus communs sont les polymorphismes nucléotidiques (SNP). Il s'agit d'une différence d'un seul nucléotide entre deux séquences d'ADN. Il a été estimé qu'il existerait environ dix millions de ces polymorphismes, observés à une fréquence plus grande ou égale à 1 % dans la population mondiale [29, 30]. Les SNP constitueraient environ 90 % de la variation chez les humains, représentant une différence à environ tous les mille nucléotides. La majorité de ces polymorphismes proviendrait d'un simple événement de mutation avec un taux de mutation de l'ordre de 10^{-8} par nucléotide par génération [31]. Lorsque le variant se retrouve

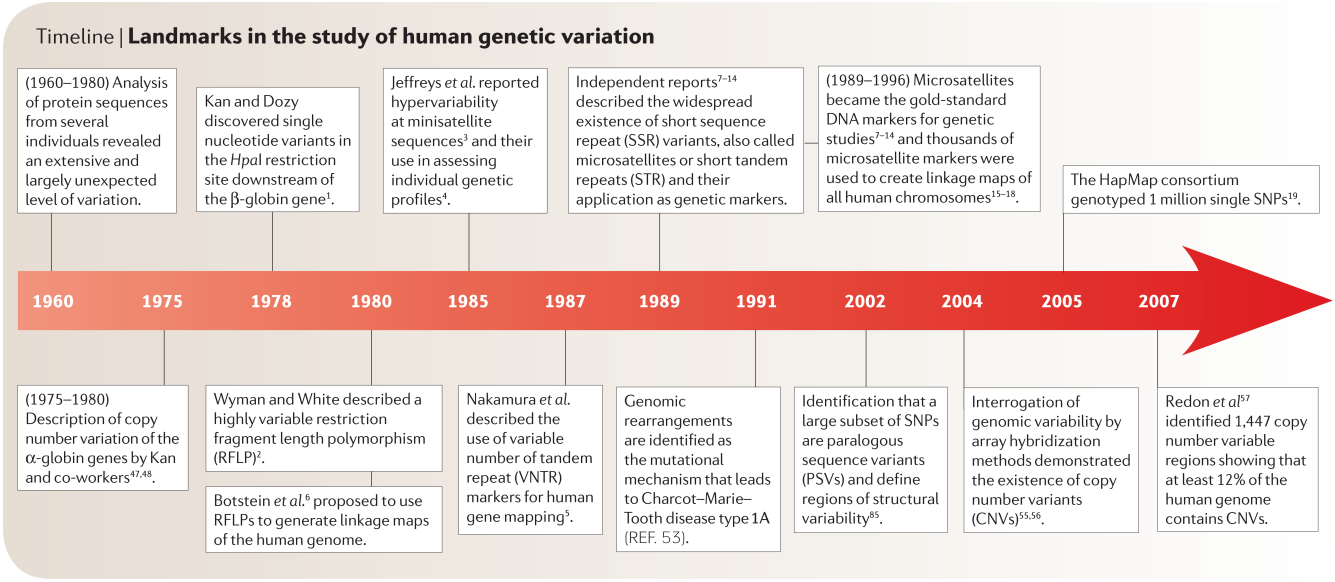
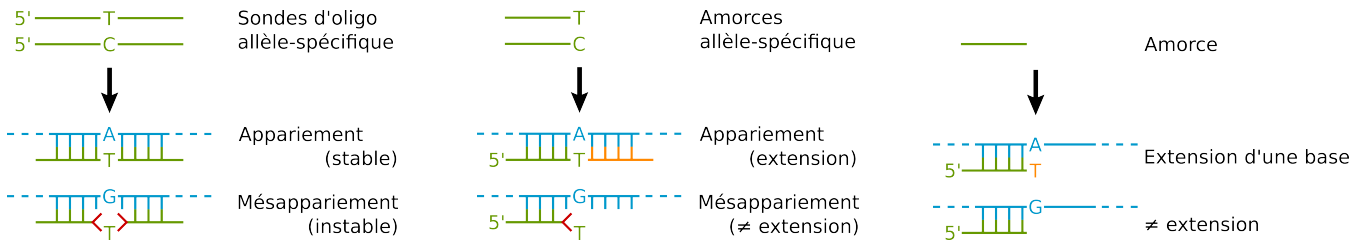


FIGURE 1.1. – Avancées dans les études de la variabilité du génome humain. Les points culminants de l'étude de la variabilité du génome furent la découverte des SNP en 1978, des microsatellites en 1989 ainsi que la démonstration de l'existence à grande échelle des CNV, en 2004. Figure adaptée de Beckmann et coll. [1].

à faible fréquence dans une population (généralement inférieure à 1 %), on parle plutôt d'une variation nucléotidiques (SNV). Les SNP sont généralement composés de deux allèles : l'allèle mineur (plus faible fréquence) et l'allèle majeur (plus haute fréquence). Ils sont habituellement exprimés par a et A, respectivement.

1.2.1. Génotypage

Plusieurs techniques permettent de déterminer le génotype de SNP dont la position est déjà connue, ou encore de faire la détection de nouveaux SNV (mutations *de novo* ou propriétaires). Ces techniques permettent d'interroger à la fois jusqu'à plusieurs millions de SNP (micropuces de SNP) ou encore de sonder le génome humain en partie ou en totalité (séquençage à haut débit). Les analyses génétiques ou statistiques ultérieures dicteront le choix de la méthode de génotypage désirée.



(a) Hybridation allèle spécifique **(b)** Extension d'amorces **(c)** Extension d'un nucléotide

FIGURE 1.2. – Principales réactions biochimiques utilisées pour le génotypage de SNP. Trois techniques sont visuellement représentées : (1.2a) l'hybridation de sonde allèle-spécifique, (1.2b) l'extension d'amorces allèle-spécifiques et (1.2c) l'extension d'un nucléotide sur une amorce. Les sondes ou les amorces sont représentées en vert, les molécules d'ADN ciblées, en bleu, et les nucléotides nouvellement synthétisés, en orange. Figure adaptée de Syvänen et coll. [2].

1.2.1.1. Micropuces de SNP

Plusieurs compagnies se sont spécialisées dans le génotypage de SNP. Certaines ont produit des techniques permettant de déterminer le génotype de plusieurs millions de SNP à la fois (p. ex. Affymetrix et Illumina), alors que d'autres permettent de choisir manuellement certains polymorphismes ou variations afin de créer des études plus spécifiques (p. ex. Sequenom). Toutes ces techniques utilisent les propriétés d'hybridation de l'ADN (Figure 1.2), mais elles se distinguent par leurs méthodes de détection du polymorphisme.

La compagnie Affymetrix utilise la technique de l'hybridation de sondes d'oligonucléotides allèle-spécifiques (Figure 1.2a). La micropuce inclut, pour tous les SNP d'intérêts, un groupe de sondes regroupant les séquences alternatives au site d'un SNP (avec l'allèle en question situé au milieu d'une sonde de 25 nucléotides). Les molécules d'ADN ciblées (fragmentées et marquées par fluorescence à l'aide d'une seule couleur) sont mises en contact avec les sondes et l'hybridation a lieu. Un algorithme est ensuite utilisé afin d'interpréter le patron complexe de fluorescence et d'assigner un génotype à chaque position testée.

La compagnie Illumina utilise une combinaison de deux méthodes différentes : l'extension d'amorces allèle-spécifique (Figure 1.2b) ainsi que l'extension d'un nucléotide sur des amorces (Figure 1.2c) [32].

La première méthode requiert deux amorces différentes, une pour chacun des deux allèles possibles, et nécessite qu'une seule couleur. La fluorescence est attachée à chacun des nucléotides (dNTP) et l'emplacement de la fluorescence marque le type d'allèle présent dans l'ADN cible [33]. La deuxième méthode requiert uniquement une amorce spécifique à la séquence en amont du SNP. La présence de nucléotides modifiés (ddNTP) permet l'extension d'une base sur l'amorce, représentant l'allèle du SNP. Cette technique ne nécessite qu'une seule amorce, mais deux couleurs différentes (afin de distinguer les deux allèles possibles du SNP) [34]. Ces deux techniques (nommées Infinium I et II par la compagnie, respectivement) résultent en une paire d'intensité pour chacun des deux allèles de tous les SNP inclus dans la micropuce. Un algorithme de *clustering* est ensuite utilisé afin de déterminer le génotype de chacun des SNP.

Il est important de mentionner que, dans les deux technologies présentées, la présence de variations ou de polymorphismes à proximité du SNP dans le génome ciblé va diminuer l'affinité de ce dernier avec la sonde ou l'amorce, ayant un impact direct sur l'intensité du signal obtenu.

1.2.1.2. Séquençage à haut débit

Le séquençage à haut débit (*high-throughput sequencing*) désigne un ensemble de plateformes récentes (depuis 2005) produisant simultanément des milliards de courtes séquences d'ADN à faible coût. Toutes ces plateformes incluent un nombre de méthodes qui peuvent être généralement regroupées en (1) la préparation d'un modèle d'ADN [1] (*template preparation*), (2) le séquençage et (3) les analyses des données générées. La combinaison unique de protocoles spécifiques est ce qui distingue une technologie d'une autre et détermine le type de données produites par chacune des plateformes. De plus, il est possible de capturer une certaine région du génome (p. ex. uniquement l'exome), évitant de faire le séquençage du génome en son entier, réduisant ainsi le coût de l'étude en cours et l'analyse bio-informatique subséquente.

1. ADN recombinant comprenant une région connue (vecteur ou adaptateur permettant l'hybridation à une sonde universelle) et un fragment du génome cible (séquence à être séquencée).

Il est très important d'utiliser une méthode robuste permettant la création d'une représentation non biaisée du génome à séquencer. Les méthodes courantes impliquent généralement une fragmentation aléatoire de l'ADN génomique en petits morceaux, permettant la création d'un modèle de fragmentation (*fragment templates*) ou un modèle d'extrémités reliées (*mate-pair templates*). Ces modèles sont ensuite immobilisés sur une surface solide ou un support quelconque. Cette immobilisation permet l'activation simultanée de centaines de milliards de réactions de séquençage. Selon la technologie choisie, le modèle est ensuite amplifié ou non (*single-molecule templates*).

Le séquençage de chacun des modèles a ensuite lieu. Une première méthode (*cyclic reversible termination* ou CRT) consiste en l'ajout d'un nucléotide fluorescent modifié permettant d'interrompre la réplication de l'ADN. Ensuite, après l'incorporation d'un unique nucléotide à chaque modèle, le restant est nettoyé et une image est prise afin de déterminer l'identité du nucléotide ajouté. Finalement, le clivage de la portion empêchant la réplication de l'ADN permet au cycle de séquençage de reprendre avec le nucléotide suivant. Les séquenceurs *HiSeq* et *MiSeq* de la compagnie *Illumina* utilisent cette méthode de séquençage. Une deuxième méthode (*ion semiconductor sequencing*) est utilisée par la compagnie *Life Technologies*. Une fois la molécule d'ADN stabilisée dans un micropuits, un détecteur d'ions capte l'ion d'hydrogène libéré par la réaction de polymérisation lors de l'incorporation d'un nucléotide au brin grandissant d'ADN. Ces nucléotides (non modifiés) sont ajoutés à la solution l'un à la suite de l'autre, permettant le séquençage en temps réel [35]. Une troisième méthode (*single-nucleotide addition* ou pyroséquençage) mesure la libération de pyrophosphate inorganique en le convertissant en lumière visible avec une série de réactions enzymatiques. En utilisant des quantités limitantes de nucléotides, ce type de séquençage permet d'arrêter la synthèse de l'ADN après l'incorporation d'un seul nucléotide par modèle. Cette technique de séquençage est utilisée par la compagnie *Roche/454 Life Sciences*. D'autres méthodes de séquençage existent, mais il ne s'avère pas pertinent d'en parler dans le cadre de ce projet ; elles sont décrites en entier par Metzker dans son article de revue de littérature [36].

Une fois les images enregistrées et analysées, l'équipe de bio-informatique a en sa possession une grande quantité de petits fragments d'ADN séquencés. Il faut ensuite les aligner sur un génome de référence ou encore en faire l'assemblage *de novo* (c.-à-d. sans génome de référence). Plusieurs outils existent à cette fin, avec leur lot d'avantages, d'inconvénients et de caractéristiques différentes [37]. Par exemple, un des problèmes connus est que ces outils possèdent tous une performance réduite lors de l'alignement de fragments se retrouvant dans des régions complexes du génome, telles que les régions répétées. Puisque ces fragments peuvent s'aligner à plusieurs endroits différents sur le génome, certains outils préféreront choisir le meilleur emplacement, choisir un emplacement aléatoire, énumérer tous les emplacements possibles ou tout simplement les éliminer. Tous les fragments éliminés ou possédant un faible score de positionnement seront difficilement utilisables lors d'étapes subséquentes, voire même inutilisables. Le choix de l'outil d'alignement dépendra donc de sa performance, du type de données d'entrée, des analyses subséquentes (c.-à-d. la détection de SNP, de SNV et de variations structurales) ainsi que des caractéristiques techniques de l'outil (p. ex. sa vitesse d'exécution, la possibilité d'aligner des fragments possédant des trous consécutifs, etc.).

De nouvelles technologies de séquençage sont en cours de développement. L'une d'elles implique le séquençage direct d'une molécule unique d'ADN (*single-molecule sequencing*). Puisque cette technologie ne requiert pas d'amplification de l'ADN au préalable, elle limite le nombre d'erreurs de séquençage dû à la polymérase lors de la préparation de la librairie tout en éliminant l'amplification préférentielle de certains fragments [38]. Les autres avantages de cette méthode incluent une augmentation du débit de séquençage, un allongement de la longueur de lecture (facilitant les assemblages *de novo*), la détection directe des haplotypes [2] ou encore le phasage [3] de chromosomes entiers [40]. Par contre, cette technologie possède certains défis à surmonter, tels que le faible rapport signal sur bruit (*signal-to-noise ratio*) et le développement de détecteurs permettant d'interpréter les faibles signaux produits lors du séquençage d'une molécule unique [38]. D'autres technologies permettront l'analyse simultanée de plusieurs propriétés d'une seule cellule, incluant la génomique, l'épigénomique, la transcriptomique et la protéomique, disciplines qui étaient contraintes à des

2. Groupe d'allèles situés sur un même chromosome et habituellement transmis en groupe.
3. Le phasage consiste à identifier les allèles qui sont co-localisés sur le même chromosome [39].

expérimentations séparées [41]. Quelles que soient les nouvelles technologies de séquençage à voir le jour d'ici les prochaines années, le travail des bio-informaticiens sera crucial et d'envergure.

1.2.2. Études déterminantes

Plusieurs études déterminantes sur les SNP et les SNV ont été effectuées. Les deux plus importantes sont celles du consortium international HapMap (*The International HapMap Consortium* [42, 43, 44]) et le projet des 1 000 génomes (*The 1000 Genomes Project* [3]). La première étude consiste en la création d'une carte d'haplotype du génome humain et implique le génotypage de plusieurs millions de SNP. La deuxième implique le séquençage à haut débit de plus de mille génomes humains afin d'identifier toutes les variations présentes (autant les SNP que les SNV et les variations structurales).

Le projet HapMap a débuté en 2002 et inclut plusieurs groupes de chercheurs à travers le monde. Le but du projet est de déterminer des patrons communs de variations génétiques chez l'humain, en caractérisant ces variations, leurs fréquences et leurs corrélations entre elles dans quelques centaines d'individus avec une descendance précise provenant de l'Afrique, de l'Asie et de l'Europe [31]. Le projet initial a été séparé en trois phases. La première phase (*phase I*) a pour but de génotyper au moins un SNP (communs à une fréquence de 0,5 % ou plus) à toutes les 5 kb à travers le génome de 269 individus provenant de populations différentes. Plus précisément, 90 individus (30 trios) de Yoruba (Nigéria, YRI), 90 individus (30 trios) de l'Utah (États-Unis, CEU), 45 Chinois Han (CHB) et 45 Japonais (Tokyo, JPT) ont été analysés afin d'obtenir leurs génotypes pour 1 007 329 SNP avec une précision de 99,7 % et une complétude de 99,3 % (après contrôle de qualité). Tous ces SNP sont polymorphiques chez les 269 individus restants. La phase I a permis de déterminer le génotype de 11 500 SNP non synonymes[4]. De tous les SNP génotypés, seulement une fraction d'entre eux était polymorphique dans une population (85 % chez YRI, 79 % chez CEU et 75 % chez CHB et JPT). Les auteurs ont remarqué que les SNP communs étaient hautement corrélés avec leurs voisins ; la densité de SNP peut être diminuée de 75 à 90 % sans perte significative

4. Mutation qui modifie la séquence en acides aminés d'une protéine.

d'information. Effectivement, l'utilisation de 260 000 (CHB et JPT) à 474 000 (YRI) SNP sont suffisants pour capturer l'information des 1 007 329 SNP initialement génotypés [42].

La deuxième partie du projet HapMap (*phase II*) visait la détermination des génotypes à plus de 4,3 millions de SNP chez les mêmes individus que ceux de la phase I. Après les différentes étapes de contrôle de qualité, un total de 3 107 620 SNP furent conservés, augmentant ainsi la densité initiale à un SNP pour toutes les kilobases. Les SNP ajoutés à la phase II sont généralement plus regroupés et possèdent une plus faible fréquence allélique que ceux de la phase I. De plus, une diminution de la corrélation inter-SNP est remarquée. Ceci est expliqué par le fait que les SNP de la phase II incluent une plus grande représentation des variations rares du génome humain. En utilisant une corrélation r^2 de 0,8 ou plus, les auteurs ont trouvé que 1,09 million de SNP sont nécessaires pour contenir toute l'information des SNP de la phase II chez les YRI (alors que seulement 500 000 sont nécessaires pour les autres populations). Ils ont tout de même observé plusieurs SNP à haute fréquence n'étant pas en corrélation avec d'autres ; ces SNP sont généralement situés dans des régions chaudes de recombinaisons (32 996 de ces régions ont été détectées) [43].

La dernière phase du projet HapMap (*phase III*) a permis d'étendre les deux premières phases en effectuant le génotypage de 1 440 616 SNP qui sont polymorphiques chez 1 184 individus provenant de 11 populations différentes, incluant tous ceux présents dans les deux premières phases du projet. Chez ces individus, les auteurs ont détecté un biais faible, mais statistiquement significatif contre les variants rares (fréquence entre 0,05 et 0,5 %), ce qui est logique compte tenu des types de technologies choisies pour le génotypage (micropuces d'Illumina et d'Affymetrix). Suite à leurs analyses des variants plus rares, les auteurs ont conclu qu'il était nécessaire de bien caractériser les paramètres génétiques populationnels dans chacune des populations et pour toutes les strates de fréquences alléliques, car il était impossible d'extrapoler à partir des études ultérieures de SNP communs. De plus, les variations plus rares étaient moins partagées à travers les différentes populations et elles possédaient de plus grands haplotypes que les variants communs. L'imputation des variations plus rares était possible, mais sous certaines conditions (larges groupes de référence, génotypage dense

et précis ainsi qu'un bon phasage) [44].

La deuxième étude de grande importance abordée ici est le *1000 Genomes Project* (1000G). Le but de cette étude est de fournir une caractérisation précise des variations du génome humain afin de faciliter l'investigation de la relation entre le génotype et les différents phénotypes. À ce jour, uniquement les données du projet pilote ont été publiées. Le projet pilote consite au séquençage à faible couverture de 179 individus (provenant des quatre populations du HapMap, 3,6× [5] par individu), au séquençage à haute couverture de deux trios (six individus à 42× chacun) et au séquençage ciblé de 8 140 exons chez 697 individus (provenant de sept populations différentes). Un total de 4,9 terabases de séquence de nucléotides d'ADN fut généré dans neuf centres de génotypage à partir d'ADN provenant de cellules lymphoblastiques immortalisées. Après les différentes étapes d'analyses, les auteurs ont identifié un total de 5,9 millions de SNP chez les deux trios, 14,4 millions de SNP chez les 179 individus séquencés à faible couverture et 12 758 SNP chez les 697 individus séquencés à haute couverture dans les exons. Une grande proportion de variants communs (90 %) était déjà connue (c.-à-d. dans la base de données dbSNP), ce qui n'était pas le cas pour les variants plus rares. Bien que les SNP communs étaient partagés à travers les différentes populations, 96 % des nouvelles variations étaient restreintes à certaines populations avec une plus grande proportion dans la population africaine ; 63 % et 44 % des SNP dans les données de basse et haute couverture pour YRI, respectivement, comparées à 33 % et 22 % dans les données de basse et haute couverture pour CEU, respectivement (Figure 1.3). Pour ce qui est des variations fonctionnelles, les auteurs estiment qu'un individu typique possède de 10 000 à 11 000 variations non synonymes et de 10 000 à 12 000 variations synonymes lorsque comparées à la référence du génome humain. Un total de 68 300 variations non synonymes furent trouvées, dont 50 % n'ont jamais été observées auparavant. Ces variations se retrouvent généralement avec une faible fréquence dans les populations étudiées. Dans les deux trios analysés, un total de 3 236 et 2 750 variations germinales *de novo* furent trouvées. Ces variations ont permis d'estimer des taux de mutation de $1{,}2\times10^{-8}$ et de $1{,}0\times10^{-8}$ pour CEU et YRI, respectivement [3].

5. Représente la profondeur de séquençage, c.-à-d. le nombre de fois, en moyenne, où un nucléotide a été séquencé.

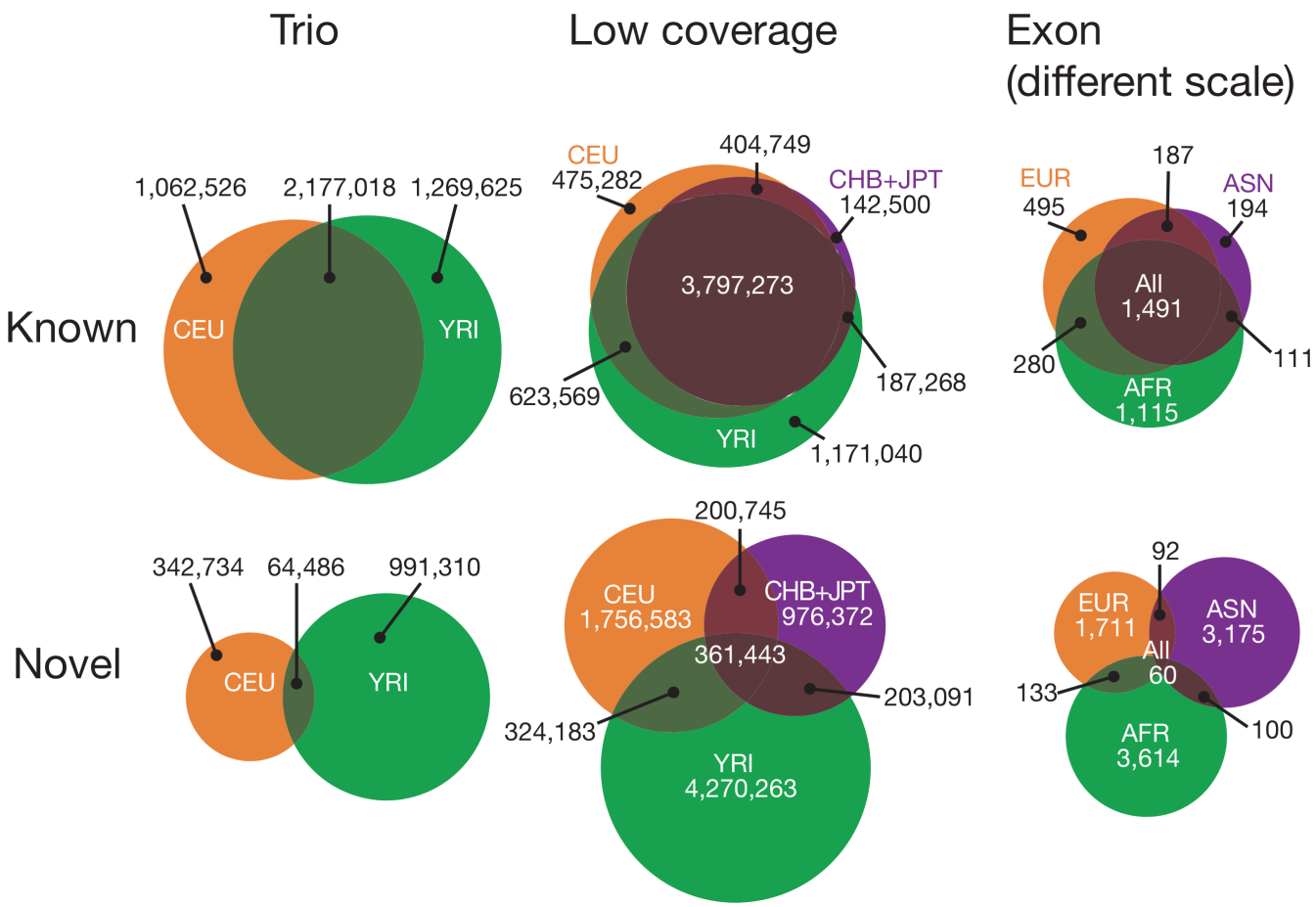


FIGURE 1.3. – Caractéristiques populationnelles des variations du *1000 Genomes Project*. Diagrammes de Venn montrant la distribution des variations connues (rangée du haut) et des variations rares (rangée du bas) issues de populations diversifiées selon le projet (les deux trios à haute couverture, les données de basse couverture sur 179 individus et les données de l'exome à haute couverture sur 697 individus). Les variations représentées incluent uniquement les SNP et les SNV. Figure tirée de l'article du *1000 Genomes Project Consortium* [3].

1.3. Polymorphismes et variations de nombre de copies

Les variations de nombre de copies (de l'anglais « *copy number variation* », CNV) sont des segments d'ADN de 1 kb ou plus qui sont présents en différent nombre de copies lorsque comparés à un génome de référence. Les polymorphismes de nombre de copies (CNP) sont des CNV dont la fréquence d'apparition dans la population est supérieure à 1 % [45]. Le terme CNV sera utilisé comme généralisation des termes CNV et CNP dans la suite du présent document, à moins d'indication contraire. Les CNV sont de plus en plus utilisés par les chercheurs afin d'étudier la relation entre les variations génomiques et les maladies. Entre 3,7 % et 12 % du génome humain pourrait être affecté par les CNV [46, 47, 48] et ceux-ci pourraient être responsables d'une quantité importante de la variabilité interindividuelle [1]. Par sa couverture génomique et sa taille, une variation en nombre de copies peut facilement impliquer un grand nombre de gènes, pouvant ainsi causer des désordres génomiques [4]. Malgré le fait que les CNV possèdent une grande héritabilité, ils peuvent apparaître de manière *de novo* à la fois dans les cellules germinales et somatiques [49]. La fréquence de mutation estimée est d'ailleurs plus élevée pour les CNV (de $1,7 \times 10^{-6}$ à $1,0 \times 10^{-4}$ par locus par génération) lorsque comparée aux SNP (de $1,8$ à $2,5 \times 10^{-8}$ par paire de bases par génération) [50].

Les CNV peuvent influencer un phénotype de différentes façons : (1) par dosage de gènes, (2) par interruption de gènes, (3) par fusion de gènes, (4) par effet de positionnement, (5) par dévoilement d'allèles récessifs ou de polymorphismes fonctionnels et (6) par effet de transvection [45, 51]. Chacun de ces mécanismes est responsable d'au moins une maladie connue [50]. Finalement, ces variations génétiques sont susceptibles de jouer un rôle important dans l'étiologie des maladies complexes et des malformations congénitales sporadiques [1], en raison de leurs hauts taux de mutation et de leur couverture génomique pouvant inclure plusieurs gènes [52]. Par exemple, un effet de positionnement est la cause d'au moins une vingtaine de maladies, comme la thalassémie alpha et la polydactylie préaxiale [53].

Il est possible que des cumulations de variations somatiques de CNV soient également impliquées dans certaines pathologies autres que le cancer, mais les études visant cette hypothèse restent à être démontrées. Il est néanmoins certain qu'une nette accumulation de CNV se produit lors de la culture cellulaire *in vitro*, ce qui peut d'ailleurs avoir un impact important sur l'usage de cette technique à des fins de conservation et d'amplification d'ADN pour la recherche [54].

1.3.1. Mécanismes d'apparition

Il y a quatre mécanismes principaux pouvant intervenir dans la création de nouveaux CNV : (1) la recombinaison homologue non allélique (*non-allelic homologous recombination* ou NAHR), (2) la jonction d'extrémités non homologues (*non-homologous end-joining* ou NHEJ), (3) le blocage de la fourche de réplication et changement de matrice (*fork stalling and template switching* ou FoSTeS) et (4) la réplication induite par une coupure de l'ADN médiée par microhomologie (*microhomology-mediated break-induced replication* ou MMBIR) [55, 50]. Le premier mécanisme, la recombinaison NAHR, produit la majorité des CNV récurrents (c.-à-d. des CNP communs chez l'humain), alors que les trois derniers (qui se déroulent lors de la réplication de l'ADN) produisent des CNV non-récurent (*de novo*) ou encore des variations structurales plus complexes [5]. Le Tableau I.I présente la distribution de duplications (79) et de délétions (131) selon leur mécanisme d'apparition respectif.

TABLE I.I. – Fréquence relative des mécanismes d'apparition des CNV. Cette fréquence est calculée à partir de 131 délétions et de 79 duplications. Ces données sont tirées de [15].

Type observés	**Récurrents**	**Non-récurrent**	**Complexe**	
Mécanismes	**NAHR**	**FoSTeS, MMBIR ou NHEJ**	**FoSTeS, MMBIR ou multiple NHEJ**	**Total**
Délétions	107 (81,7 %)	21 (16,0 %)	3 (2,3 %)	131
Duplications	56 (70,9 %)	9 (11,4 %)	14 (17,7 %)	79

1.3.1.1. Recombinaison homologue non allélique

La recombinaison NAHR est l'un des premiers mécanismes responsables de désordres génomiques à avoir été identifié. Elle est responsable de la majorité des CNV récurrents (c.-à-d. communs) et peut avoir lieu à la fois lors de la méiose ou de la mitose [50]. La récurrence est médiée par une

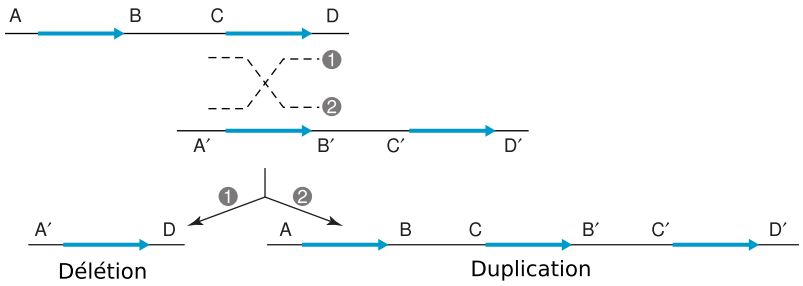


FIGURE 1.4. – Recombinaison homologue non allélique (NAHR). Réarrangement génomique résultant d'une recombinaison entre séquences répétées (duplications segmentales, flèches bleues). Les lettres majuscules représentent des séquences flanquantes uniques (non dupliquées). Les lignes pointillées représentent l'endroit où la recombinaison a lieu. Puisque l'orientation des séquences répétées est la même, le résultat de la recombinaison est à la fois une duplication et une délétion. Figure adaptée de Lupski et coll. [4].

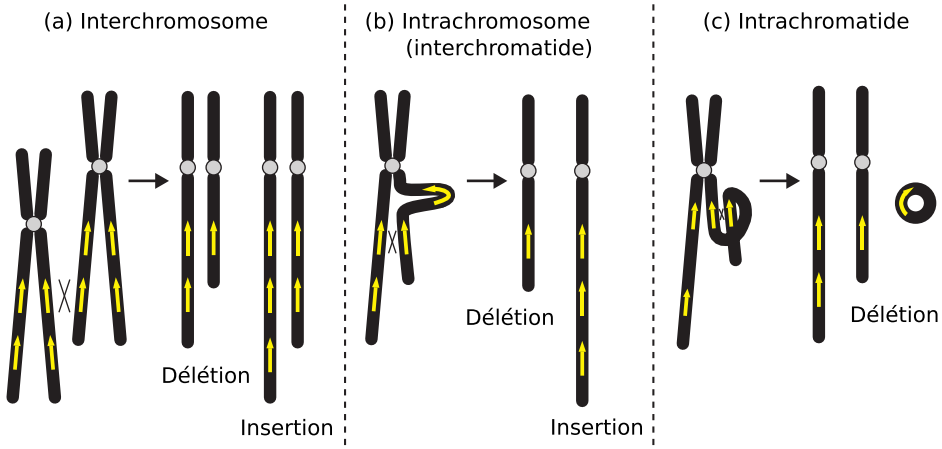


FIGURE 1.5. – Production de délétions et d'insertions par NAHR. Les recombinaisons NAHR produisent des délétions et des insertions de trois manières : croisement interchromosomique, intrachromosomique ou intrachromatidale. Ce dernier ne produit que des délétions. Figure adaptée de Liu et coll. [5].

structure (ou architecture) génomique commune où les régions modifiées en nombre de copies (ou en orientation) sont flanquées par des séquences répétées paralogues (duplications segmentales ou *low-copy repeats*) de haute identité (>97 %) et d'une longueur variant de 10 à 400 kb [56]. Lorsqu'une recombinaison a lieu entre des duplications segmentales orientées dans la même orientation, le résultat obtenu est à la fois une délétion ainsi qu'une duplication (voir Figure 1.4). Lorsque l'orientation est inversée, le résultat obtenu est une inversion de la région située entre les duplications segmentales.

Le mécanisme de recombinaison NAHR favorise les délétions sur les duplications, dépendamment de sa localisation (inter ou intrachromosomique) ; les délétions peuvent avoir lieu *in cis* et *in trans*, alors que les duplications peuvent uniquement avoir lieu *in trans* (Figure 1.5). Le ratio de délétion sur duplication est généralement de l'ordre de 2 :1 [57, 15]. La fréquence d'occurrence de recombinaisons NAHR est hautement corrélée à la longueur des duplications segmentales et est fortement influencée par la distance les séparant.

1.3.1.2. Jonction d'extrémités non homologues

La jonction d'extrémités non homologues (NHEJ) joue un rôle dans les cellules humaines permettant de réparer les coupures double-brin de l'ADN causées par de la radiation ionisante ou des espèces d'oxygène réactif [50]. Contrairement à la recombinaison NAHR, le NHEJ ne nécessite pas de régions d'homologies étendues et laisse une trace dans le génome sous la forme d'une perte de plusieurs nucléotides à l'endroit de jonction [58]. Les CNV produits par ce mécanisme ne sont pas récurrents (c.-à-d. extrémités et localisation différentes dans une population) [55].

1.3.1.3. Blocage de la fourche de réplication et changement de matrice

Ce mécanisme peut être à l'origine de réarrangements complexes, de duplications et de délétions non récurrents. Selon Lee et coll., lors de la réplication de l'ADN, la fourche de réplication peut se bloquer dans des régions d'instabilités génomiques ou près de régions de duplications segmentales. Le brin indirect (ou *lagging strand*) peut alors se désengager et envahir une deuxième fourche de réplication active en utilisant des régions de microhomologies (Figure 1.6A). Après un certain temps, le brin nouvellement créé peut se dégager une autre fois et envahir une troisième fourche de réplication active (Figure 1.6C). Éventuellement, la fourche de réplication initiale terminera sa tâche normalement (Figure 1.6D) [6, 59, 60]. Selon la localisation des fourches de réplications secondaires relativement à la fourche principale, soit en aval ou en amont, le changement de matrice peut produire une délétion ou une duplication, respectivement [50]. Cette série de désengagements et d'invasions du brin indirect peut se produire plusieurs fois.

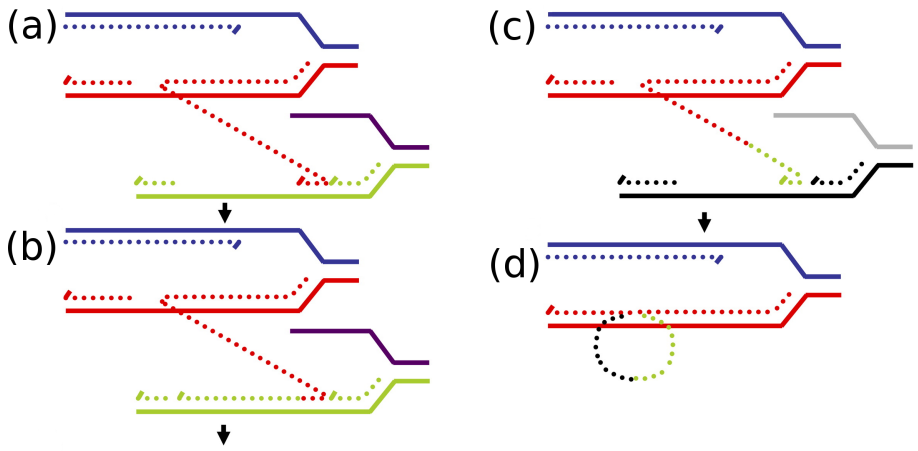


FIGURE 1.6. – Mécanisme du FoSTeS. (a) Après avoir rencontré une lésion de l'ADN, une fourche de réplication (lignes solides bleues et rouges) peut voir son brin indirect (lignes pointillées rouges) se désengager et envahir une seconde fourche de réplication active (lignes solides mauves et vertes) et (b) poursuivre la synthèse de l'ADN (ligne pointillée verte). (c) Un second désengagement peut avoir lieu : le brin indirect peut envahir une troisième fourche de réplication (lignes solides grises et noires). (d) Finalement, le brin indirect retourne sur la matrice originale. Figure adaptée de Lee et coll. [6].

1.3.1.4. Réplication induite par une coupure de l'ADN médiée par microhomologie

Lors de la réplication, la fourche de réplication peut s'effondrer suite à une coupure dans l'ADN. Un mécanisme de réplication induit par cette coupure (*break-induced replication* ou BIR) permettra de redémarrer la réplication et de la terminer (Figure 1.7i). Comme dans le mécanisme FoSTeS, il peut arriver que le brin nouvellement formé envahisse une autre région chromosomique partageant une certaine homologie [59] (Figure 1.7ii). Les conséquences de ce changement de matrice lors de la réparation de la coupure incluent les duplications et les délétions, selon que l'invasion soit située après ou avant la localisation de la coupure sur la région homologue [7]. Ce mécanisme se nomme MMBIR (*microhomology-mediated break-induced replication*).

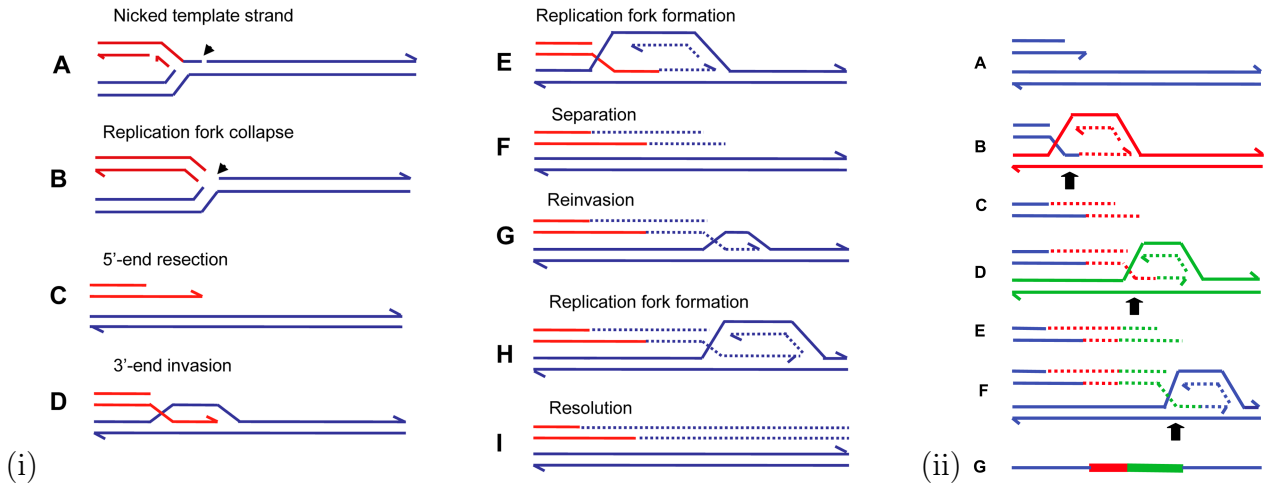


FIGURE 1.7. – Réplication induite par une coupure de l'ADN (BIR) médiée par microhomologie (MMBIR). Une fourche de réplication peut s'effondrer lorsqu'elle rencontre une coupure dans l'ADN (iA-B). Le mécanisme BIR (i) permettra de réparer cette coupure et de compléter la réplication. Une extrémité cohésive est alors créée (iC) et le segment simple brin va ensuite envahir le brin en cours de réplication (iD), permettant la synthèse de l'ADN (iE). Ces étapes seront recommencées jusqu'à ce que la réplication soit terminée (iF-I). Il peut arriver que des régions de microhomologies sur des matrices différentes (p. ex. chromosome homologue) se retrouvent à proximité. Le segment simple brin peut alors envahir cette région et ainsi produire des réarrangements génomiques (iiB-G). Figure adaptée de Hastings et coll. [7].

1.3.2. Génotypage

Depuis le début des années 2000, le développement de stratégies expérimentales et computationnelles a permis l'analyse des variations structurales du génome humain avec une meilleure résolution qu'auparavant. La majorité de ces méthodes nécessite une comparaison avec un ou des génomes de référence.

1.3.2.1. Hybridation génomique comparative

Lors d'une hybridation génomique comparative (CGH) [61], deux échantillons d'ADN sont extraits, l'un provenant de l'individu testé et l'autre, de l'individu de référence. Les deux échantillons sont marqués à l'aide de deux couleurs fluorescentes différentes. Les échantillons sont ensuite appliqués à une représentation du génome (une librairie, c.-à-d. des chromosomes en métaphase) et compétitionnent pour des sites d'hybridations complémentaires. Les duplications et les délétions sont

représentées par une augmentation et une diminution du ratio de fluorescence [6], respectivement. Un des avantages de cette méthode est l'absence de connaissance *a priori* des débalancements chromosomiques impliqués. Par contre, il est impossible de détecter d'autres types de réarrangements balancés, tels que les inversions et les translocations. De plus, aucune information n'est disponible à savoir si la variation trouvée est présente chez l'individu test ou de référence. La résolution minimale de la technique CGH est d'environ 10 Mb pour les gains et les pertes d'un petit nombre de copies [62] et d'environ 2 Mb pour les amplifications de plusieurs copies [63].

Afin d'obtenir une meilleure résolution et une cartographie directe des aberrations sur la séquence de référence du génome humain et afin d'utiliser cette technologie pour des études de masse, l'utilisation de puces est recommandée. Les puces CGH (*array-CGH*) utilisent des clones afin de remplacer les chromosomes en métaphase. Il existe plusieurs types de plateformes et elles utilisent généralement des clones provenant de chromosomes bactériens artificiels (BAC), d'ADN complémentaire ou d'oligonucléotides [64]. La résolution des puces est restreinte seulement par la taille des clones utilisés ainsi que par la densité de ces derniers sur la puce. Un désavantage de cette méthode est que la taille des variations trouvées est souvent surestimée, car on associe souvent cette dernière à la taille du clone, ce qui est une supposition erronée [65]. La meilleure résolution est obtenue en utilisant des oligonucléotides. Ces séquences sont sélectionnées à partir du génome de référence humain afin d'obtenir la meilleure couverture possible [66]. Les logiciels permettant d'automatiser l'analyse des ratios de fluorescence sont, entre autres, *aCGH* [67], *CLAC* [68], *CBS* [69, 70] et *GLAD* [71].

Il existe plusieurs variantes des puces CGH. Fiegler et coll. ont choisi manuellement des fragments du génome de référence et les ont validés en séquençant leurs extrémités afin de créer une puce contenant près de 26 500 clones et couvrant environ 93,6 % du génome de référence. Cette puce est nommée *Whole-Genome Tiling Path* (WGTP) [72]. Un autre type de puce CGH est la *Representational Oligonucleotide Microarray Analysis* (ROMA). Cette variante diffère des autres par la méthode de préparation de l'ADN. Elle utilise une représentation de l'ADN obtenue en réduisant sa complexité à l'aide d'une ou de plusieurs enzymes de restriction. Ensuite, des conditions spéciales de PCR sont

6. Logarithme en base 2 de l'intensité test sur l'intensité de référence.

utilisées afin d'obtenir des fragments d'environ 1,2 kb. Il est estimé qu'environ 200 000 fragments d'ADN sont ainsi amplifiés, couvrant approximativement 2,5 % du génome humain avec un *signal-to-noise* amélioré [73].

1.3.2.2. Puces de SNP

Les puces de SNP peuvent être utilisées afin de trouver des variations structurales. Les deux puces les plus couramment utilisées proviennent d'Illumina et d'Affymetrix. Les intensités d'hybridation sont comparées à des valeurs moyennes dérivées d'un ensemble de témoins [45]. Plusieurs algorithmes existent pour la détection de régions variant en nombre de copies et la majorité est spécifiquement conçue pour un type de puce. Par exemple, *Genotyping Console* (logiciel propriétaire) et *Birdsuite* [65] peuvent être utilisés pour les puces Affymetrix, alors que *PennCNV* [74] et QuantiSNP [75] le sont pour les puces Illumina. La majorité de ces algorithmes utilisent des modèles de Markov cachés (de l'anglais « *hidden Markov models* », HMM) pour effectuer la détection. Les algorithmes utilisent deux types de métriques afin de faire le génotypage des CNV : le *log R ratio*[7] (LRR) et le *B allele frequency*[8] (BAF) [76] pour *QuantiSNP* et *PennCNV* ou bien simplement l'intensité brute pour *Birdsuite* et *Genotyping Console*. Le choix de la méthode utilisée dépendra de plusieurs facteurs : la plateforme utilisée (Illumina ou Affymetrix), le choix du type de CNV détecté et le type d'ADN analysé [77].

Bien que les puces de SNP apportent une très grande couverture du génome, certaines régions, généralement celles contenant des variations structurales, se manifestent par une diminution de la mesure de densité des sondes [78]. Par exemple, bien qu'une forte association des CNV avec les duplications segmentales ait été démontrée [79, 80], ces régions présentent toujours une plus faible densité de marqueurs due à la difficulté à créer des sondes spécifiques à ces régions [81].

7. Logarithme normalisé de l'intensité, soit $\log_2(R_{\text{subject}}/R_{\text{expected}})$ où R_{subject} et R_{expected} est l'intensité normalisée du sujet et attendue, respectivement.

8. Ratio d'intensité d'allèles. Des ratios de 0, 0,5 et 1 représentent des génotypes homozygote A, hétérozygote AB et des homozygote B, respectivement.

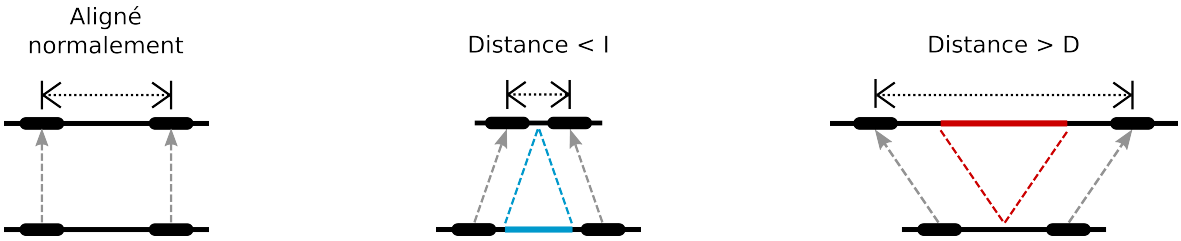
1.3.2.3. Méthodes algorithmiques

Certaines méthodes utilisent des techniques de type algorithmique afin d'identifier et de caractériser des variations structurales à petite échelle. Elles tirent parti des méthodes de détection des SNP dans les populations humaines. Ces méthodes se basent sur le fait que la présence d'une délétion laisse une empreinte visible sur les SNP la composant [82, 83]. Il est possible de détecter des délétions en cherchant des ensembles de SNP consécutifs présentant une déviation apparente de transmission mendélienne ou de l'équilibre d'Hardy-Weinberg ou encore, présentant des génotypes nuls [84]. En revanche, il est plutôt difficile de détecter des duplications avec ces méthodes.

1.3.2.4. Séquençage à haut débit

Il est possible d'identifier des variations structurales à l'aide des techniques de séquençage de nouvelle génération ; toute variation structurale devrait laisser une signature quelconque dans les données de séquençage [85]. Les deux méthodes les plus couramment utilisées sont le séquençage d'extrémités appariées (*paired-end sequencing*) et l'analyse de la profondeur de lecture (*read-depth analysis*).

Afin de pouvoir séquencer efficacement les extrémités de longs fragments aléatoires d'ADN, une technique consiste en la circularisation de ces fragments. Pour ce faire, deux méthodes peuvent être utilisées. La première consiste en la création d'une librairie génomique contenant une grande quantité de clones de taille constante. Le *fosmid* est le vecteur par excellence, puisqu'il assure une taille d'environ 40 kb [86]. Par la suite, il suffit de digérer les vecteurs à l'aide d'une enzyme de restriction et de séquencer chaque côté du clone [87, 88]. La deuxième méthode consiste à faire une fragmentation aléatoire de l'ADN génomique afin d'obtenir une taille moyenne et constante de fragments. Par la suite, il faut protéger les fragments contre l'enzyme de restriction choisie à l'aide de la méthylation, ajouter des adaptateurs aux fragments et les digérer avec l'enzyme de restriction. Les extrémités cohésives des adaptateurs permettront de circulariser le fragment à séquencer. Après avoir coupé l'ADN circularisé de manière aléatoire et après avoir ajouté d'autres adaptateurs, il suffit de séquencer la molécule en son entier ou à ses deux extrémités [8]. Pour rendre possible la détection



(a) Normal **(b)** Insertion **(c)** Délétion

FIGURE 1.8. – Découverte de CNV à partir d'extrémités appariées. Lorsque la distance apparente sur le génome de référence (lignes du haut) est la même que la distance attendue entre les extrémités appariées (lignes du bas), il y a un nombre normal de copies (1.8a). Lorsque cette distance est plus petite qu'un seuil I, il y a insertion (1.8b). Finalement, lorsque la distance est plus grande qu'un seuil D, il y a délétion (1.8c). Figure adaptée de Korbel et coll. [8].

des variations structurales, telles que les insertions, les délétions et les inversions, il faut aligner les extrémités appariées sur la séquence de référence du génome humain à l'aide d'un algorithme d'alignement, tel que *megablast* [89], BWA [90, 91], Bowtie2 [92] ou une variante de l'algorithme de Smith-Waterman [93]. Pour les insertions et les délétions, il faut chercher des extrémités appariées dont la distance sur le génome de référence (distance apparente) est significativement différente de la distance réelle sur le génome test. Les régions où la distance apparente est plus petite qu'un seuil I (Figure 1.8b) ou plus grande qu'un seuil D (Figure 1.8c) représentent respectivement les délétions et les insertions putatives. Lorsque l'orientation des extrémités appariées semble inconsistante entre le génome de référence et le génome de test, il y a présence d'une inversion putative. À l'aide de cette méthode, il est généralement possible de détecter les variations structurales d'une longueur supérieure à 8 kb [88]. De plus, la présence simultanée d'extrémités concordantes et discordantes semble prédire un état hétérozygote. Par contre, cette méthode implique la connaissance de la distribution de la longueur des régions insérées (c.-à-d. la distance entre les extrémités appariées).

La deuxième méthode de détection de variations structurales à partir des résultats de séquençage est l'analyse de la profondeur de lecture. Les séquenceurs de nouvelle génération produisent plusieurs lectures (*reads*) d'une longueur variant entre 26 et 964 paires de bases [36]. Puisque chacune de ces lectures représente un fragment aléatoire du génome test et que plusieurs fragments sont séquencés plusieurs fois, la densité de ces lectures variera à travers le génome. Une fois comparées à la

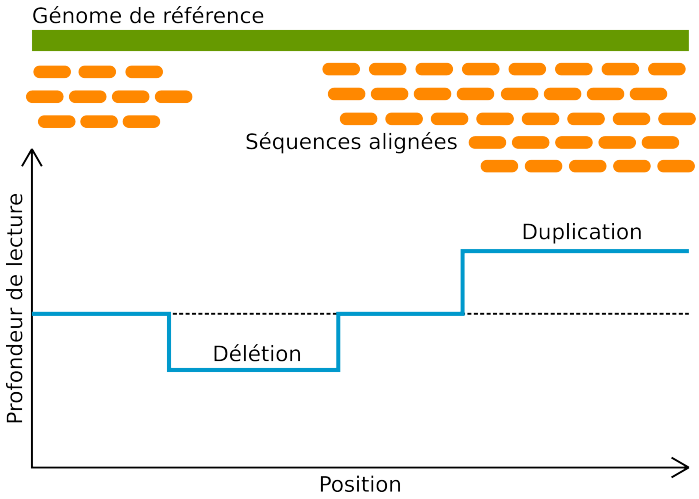


FIGURE 1.9. – Découverte de CNV à partir de la profondeur de lecture. Utilisation de la profondeur de lecture du séquençage à haut débit afin de détecter des CNV et des CNP. Cette détection est possible par une différence significative de la profondeur de lecture relativement à la profondeur de lecture attendue lors de l'alignement au génome de référence.

densité de lecture moyenne à travers le génome, les variations significatives de la densité de lecture permettront de détecter des régions présentant des duplications ou des délétions (Figure 1.9) [94]. Une limite de cette méthode est que la densité de lecture moyenne peut être grandement affectée dans les régions de fortes homologies telles que les duplications segmentales ou les familles de gènes. Une diminution de la profondeur de lecture ne veut donc pas nécessairement dire qu'il y a présence d'une délétion, mais que cette région pourrait être difficile à séquencer. Les méthodes apportant une correction à la profondeur de lecture (p. ex. un PCA [9], tel que proposé par Fromer et coll. [95]) ou encore un algorithme hybride utilisant à la fois la profondeur de lecture et les extrémités appariées auront donc un net avantage.

Finalement, il est important de mentionner que les algorithmes permettant de déterminer le génotype des CNV dans des données de séquençage à haut débit utiliseront une méthodologie distincte, compte tenu du type de séquençage ciblé (c.-à-d. tout le génome ou seulement l'exome).

9. Analyse par composante principale.

1.3.3. Études déterminantes

Plusieurs études antérieures portant sur la découverte de variations structurales (insertions, duplications, délétions ou inversions de séquences d'ADN) existent. Ces études, correspondant aux études les plus marquantes depuis 2004, utilisent les méthodes de détections énumérées précédemment. Nous présentons ici, les études ayant le plus marqué l'avancé des connaissances et des techniques dans le domaine.

1.3.3.1. Abondance des CNP chez des individus sains

Deux études déterminantes ont permis de démontrer pour la première fois l'abondance des variations en nombre copies chez des sujets sains. En 2004, Sebat et coll. [28] ont utilisé la technologie ROMA afin de détecter les variations en nombre de copies entre différents individus normaux. Ils ont segmenté l'ADN en petits fragments afin d'obtenir une sonde à tous les 35 kb environ. Les auteurs ont identifié 71 CNP chez les 20 individus analysés. Ils ont estimé une moyenne de 11 CNP entre deux individus non reliés, avec une longueur moyenne de 465 kb. Les CNP putatifs semblaient distribués à travers le génome, mais certaines régions en présentaient une plus grande densité. Les CNP étaient généralement localisés près de duplications segmentales. Il y avait 70 gènes se retrouvant dans une région variant en nombre de copies. Les auteurs ont estimé que les longs CNP ($\geq$ 100 kb) contribuaient substantiellement à la variation génomique entre individus normaux et que ces longues régions pouvaient contenir plusieurs gènes. Par contre, il est important de mentionner que les auteurs n'ont étudié qu'une petite partie du génome humain avec une faible résolution, ce qui rend pratiquement impossible la détermination des extrémités des CNP et donc, de la proximité de gènes. L'étude de Sebat et coll. fut la première à démontrer l'existence à grande échelle de CNP chez des individus sains.

Aussi en 2004, Iafrate et coll. [27] ont eu recours à une puce CGH utilisant des BAC avec une résolution moyenne d'un clone à tous les 1 Mb, correspondant à une couverture d'environ 12 % du génome humain. Ils ont étudié 55 individus non reliés, soit 39 témoins d'origine caucasienne, amérindienne, chinoise, indo-pakistanaise et africaine subsaharienne et de 16 individus présentant

des déséquilibres chromosomiques. Ils ont identifié 255 clones génomiques présentant un gain ou une perte entre les individus testés. Ces clones semblaient distribués de manière aléatoire à travers le génome. En moyenne, il y avait présence de 12,4 grandes variations en nombre de copies par individu. Certaines de ces variations étaient très grandes (2 Mb). La majorité des régions trouvées (41 %) apparaissaient chez plus d'un individu, dont 24 chez plus de 10 % de la population étudiée. Un total de 26 CNP se retrouvaient dans une région chevauchant des duplications segmentales et 13 se retrouvaient à proximité de trous présents dans l'assemblage de référence du génome humain. Au moins 67 clones englobaient complètement un ou plusieurs gènes et 14 CNP étaient localisés près de régions associées à des syndromes génétiques. Puisque les clones étaient longs, il était impossible de déterminer efficacement les extrémités des CNP putatifs, surestimant ainsi le nombre de gènes englobés par les CNP. Malgré la faible couverture du génome par la technologie de détection utilisée, Iafrate et coll. ont prouvé que de longs CNP pouvaient se retrouver chez des sujets sains, avec une fréquence aussi haute que 20 % dans les populations étudiées.

1.3.3.2. Approches algorithmiques

Deux études ont démontré qu'il était possible de détecter les variations en nombre de copies uniquement à l'aide de méthodes algorithmiques et en utilisant des données déjà existantes. En 2006, Conrad et coll. [83] ont utilisé une telle approche afin de détecter des délétions chez deux ensembles de 30 trios provenant de la population du HapMap (YRI et CEU). Pour ce faire, ils ont analysé plus d'un million de SNP du HapMap afin de trouver des ensembles de marqueurs consécutifs présentant des inconsistances mendéliennes compatibles avec une délétion chez les parents qui sont ensuite transmises à l'enfant. Ils ont obtenu un total de 228 et 396 délétions putatives distinctes pour CEU et YRI, respectivement. Ces délétions avaient une longueur entre 0,3 et 404 kb et entre 0,5 et 1 200 kb pour CEU et YRI, respectivement. Environ 16 % des délétions contenaient ou chevauchaient des régions de duplications segmentales. La plupart des délétions (39 %) étaient détectées chez un seul trio HapMap. Un total de 37 des régions putatives distinctes était partagé entre les deux populations. Il y avait un faible taux de SNP génique [10], soit 23,6 % et 18,6 % des SNP issus des populations YRI et CEU, respectivement. Il y avait tout de même couverture de 267

10. SNP présent dans un gène (intron ou exon).

gènes au total.

Aussi en 2006, McCarroll et coll. [84] ont utilisé une approche systématique afin d'identifier les délétions à partir de patrons d'échec mendélien, de déviation apparente de l'équilibre d'Hardy-Weinberg (généralement causée par le manque de SNP hétérozygote) ou encore de génotypes nuls (causés par une délétion homozygote). Pour ce faire, ils ont regardé l'ensemble des SNP du HapMap (1,3 million de SNP) afin de trouver des régions du génome présentant un même profil d'échec à des SNP consécutifs d'une manière statistiquement inattendue par chance [11]. Les quatre populations du HapMap ont été étudiées. Les auteurs ont trouvé 541 délétions candidates d'une longueur allant de 1 kb à 745 kb. Un total de 120 délétions était homozygote et 51 % se retrouvaient chez plusieurs individus non reliés. La majeure partie de ces délétions étaient nouvelles (94 %). Ceci est probablement dû à l'augmentation de la résolution par les SNP du HapMap et par le fait que la majeure partie des variations trouvées étaient de petite taille. Pour ce qui est de la localisation de ces délétions, 4,4 % étaient entourées par des duplications segmentales, alors que 0,6 % était attendue par chance. Un total de 266 gènes étaient affectés par des délétions. Les auteurs ont observé un fort déséquilibre de liaison (LD) entre ces régions validées et les SNP avoisinants. Ce qui distingue cette étude de celle de Conrad et coll. [83] est l'utilisation non seulement des inconsistances mendéliennes, mais aussi de la présence d'ensemble de génotypes nuls ou SNP présentant une déviation apparente de l'équilibre d'Hardy-Weinberg. Cet ajout permet d'augmenter la chance de détecter de nouveaux CNP. Cette étude fut aussi l'une des premières à discuter de la présence d'un déséquilibre de liaison entre CNP et SNP.

1.3.3.3. Utilisation des micropuces de SNP

En 2006, Redon et coll. [46] ont utilisé deux technologies différentes, soit une puce de SNP (Affymetrix 500K EA, 474 642 SNP, utilisant l'algorithme de Komura et coll. [96]) ainsi qu'une puce CGH (WGTP, 26 574 clones représentant 93,7 % du génome). Les 270 individus du HapMap ont été analysés. Les auteurs ont retrouvé un total de 1 447 CNV distincts, couvrant 12 % du génome humain (360 Mb). La taille moyenne de ces variations était de 341 kb et 206 kb pour

11. Probabilité binomiale d'observer chacun de ses patrons n fois pour m marqueurs.

les puces WGTP et 500K EA, respectivement. Les délétions semblaient trois fois plus courtes que les duplications, mais aucune différence significative n'a été observée en ce qui a trait à leur nombre. Un total de 30 % des régions trouvées chevauchait des variations caractérisées par des études antérieures. La proportion d'un chromosome couvert par un CNV variait entre 6 % et 19 %. Les trous dans l'assemblage de référence du génome humain semblaient encore une fois être associés aux CNV trouvés. Une proportion de 24 % des CNV était associée à des duplications segmentales. Bien que les CNV semblaient préférentiellement situés hors des gènes et des éléments ultraconservés du génome, 58 % des CNV chevauchaient de telles régions. Environ 51 % des CNV trouvés étaient en forte corrélation ($r^2 > 0{,}8$) avec les SNP avoisinants. Ces chiffres sont toutefois possiblement inexacts, étant donné la surestimation de la taille des CNV due à la taille des clones utilisés. Le LD avec les SNP diminuait grandement pour les CNV présentant plusieurs formes. Redon et coll. furent parmi les premiers à utiliser une puce conçue pour le génotypage de SNP afin de procéder à la détection à grande échelle des CNP dans une large population d'individus.

En 2008, McCarroll et coll. [97] ont conçu et utilisé une puce de SNP Affymetrix 6.0, composée de 906 600 sondes de SNP et de CNV. L'algorithme permettant la détection des CNV à partir des données d'intensités brutes provenant des 270 individus du HapMap était *Birdseye* [65]. Les auteurs ont ainsi identifié 3 048 régions de CNV, dont 60 % chevauchaient des régions reportées ultérieurement. Selon leurs résultats, les auteurs ont estimé que les CNV supérieurs à 50 kb affectaient moins de 5 % du génome chez les 270 individus sains du HapMap. Environ la moitié (1 292) des régions était observée chez plusieurs individus non reliés (plusieurs avec une MAF $\geq$ 5 %). La plupart de ces CNP (89 %) possédaient deux allèles alors qu'une petite proportion (10 %) en présentaient plusieurs. Les CNP possédant deux allèles présentaient un faible taux de déviation de l'héritabilité mendélienne et étaient compatibles avec l'équilibre d'Hardy-Weinberg. La plupart de ces CNP étaient parfaitement capturés par au moins un tag SNP du HapMap. Par contre, la distribution de r^2 était un peu plus faible que pour les SNP, provenant probablement de la pauvreté des SNP du HapMap dans les régions susceptibles de présenter des CNP. Il est à noter ici que seulement les CNP avec une MAF supérieure à 5 % ont été analysés pour le LD et

qu'aucune information n'était disponible pour les CNV et les variations complexes. Les auteurs ont aussi identifié 82 % des régions de Redon et coll. [46], mais ici aussi, les régions trouvées étaient généralement entre 5 et 10 fois plus petites. McCarroll et coll. ont conçu l'une des premières puces de SNP intégrant des sondes permettant de mesurer efficacement le nombre de copies d'environ 1,8 million de régions distribuées uniformément à travers le génome.

1.3.3.4. Études à haute résolution

En 2010, Conrad et coll. [48] ont utilisé un ensemble de 20 puces CGH *NimbleGen* comprenant un total de 2,1 millions de sondes chacune et présentant un espacement d'environ 60 paires de bases. Ces puces étaient capables de déceler des CNV d'une longueur minimale de 500 pb. L'hybridation a été réalisée pour 40 individus sains (19 femmes CEU, 20 femmes YRI ainsi que l'individu *NA15510*[12]) et l'individu de référence *NA10851*. Cette puce a permis la découverte de 11 700 CNV putatifs d'une longueur variant de 443 pb à 1,28 Mb. Environ la moitié de ces CNV (49 %) étaient présents chez un individu unique. Afin de générer le génotype de 92 % des CNV trouvés avec le premier ensemble de puces, les auteurs ont utilisé une puce CGH d'*Agilent* et une puce de SNP d'*Illumina*. Après avoir utilisé des seuils de détection très stricts et après avoir validé les résultats par qPCR et par trois puces différentes (*NimbleGen*, *Agilent* et *Illumina*), ils ont réduit leur ensemble de CNV à 8 599, représentant 77 % de délétions, 16 % de duplications et 7 % possédant plusieurs allèles. Ces CNV couvraient 3,7 % du génome humain. De plus, 38,8 % des CNV validés couvraient 13,4 % des gènes *RefSeq*, altérant la structure des transcrits (12,5 % des gènes), la séquence codante (5,5 % des ARNm) ou le cadre de lecture ($>$ 50 % des gènes). Ils produisaient une perte totale d'allèles pour 267 gènes. Pour ce qui est de la corrélation entre les SNP avoisinants et les CNV, elle était très faible pour les CNV rares et plus forte pour les délétions que les duplications. En tout, 77 % des CNV montraient une forte corrélation (r^2) avec les SNP avoisinants. L'étude de Conrad et coll. a permis pour la première fois la détection de petits CNV (aussi petits que 500 pb). Ils ont aussi déterminé que la longueur des CNV dépendait du mécanisme de mutation, les plus longs CNV étant produits par NAHR.

12. *Polymorphism Discover Resource Individual.*

En 2010, Park et coll. [94] ont proposé une méthodologie intégrant les puces CGH avec le séquençage de nouvelle génération [13] afin d'étudier trente femmes asiatiques et deux individus dont le génome a été séquencé, soit un homme et une femme coréens (*AK1* et *AK2*, respectivement). L'individu de référence pour la puce CGH était *NA10851* et a aussi été séquencé. La puce CGH contenait 24 millions de sondes permettant la détection de CNV aussi petits que 438 pb. Afin d'optimiser la concordance entre la puce CGH et l'information de séquençage, l'information sur la profondeur de lecture de *AK1* et *NA10851* a été utilisée. Les auteurs ont obtenu un total de 21 905 segments de CNV dont environ la moitié de ceux-ci, identifiée par la puce CGH pour *AK1* et *AK2*, présentait un nombre de copies de 2 suite à l'analyse de la profondeur de lecture de ces deux individus (l'individu de référence présentait un nombre différent). Aussi, 10 980 segments de CNV ont pu être mal identifiés, puisque l'individu *NA10851* présentait un nombre de copies différent de 2 dans ces régions. Finalement, 3 164 segments de CNV *secrets* [14] ont été identifiés. Le séquençage des trois individus (*AK1*, *AK2* et *NA10851*) a permis d'obtenir l'état absolu en nombre de copies pour un total de 20 099 segments de CNV. De ceux-ci, 3 164 étaient *secrets* et 6 010 étaient *obscures* [15]. La proportion de délétion était de 72,6 %, ce qui est compatible avec les études antérieures. Les longueurs moyennes des délétions et des duplications étaient respectivement de 11,8 kb et 30,3 kb. Encore une fois, les gains de copies étaient plus fréquents que les pertes dans les régions géniques. Un total de 670 CNV (3 %) était retrouvé chez chacun des individus asiatiques étudiés, couvrant 11,21 Mb et chevauchant 389 gènes par personne. En regroupant les CNV qui se chevauchaient à plus de 50 %, les auteurs ont obtenu 5 177 éléments de CNV (CNVE) couvrant 3,3 % du génome. Ces régions chevauchaient 2 913 gènes *RefSeq* et 1 483 gènes *OMIM*. L'avantage de la méthode de Park et coll. est l'utilisation simultanée de puces aCGH à très haute densité et du séquençage à haute couverture, rendant possible la détection de CNV aussi petits que 438 pb.

1.3.3.5. Déséquilibre de liaison avec les SNP

En 2010, Le *Wellcome Trust Case Control Consortium* (WTCCC) [99] a créé une puce aCGH personnalisée afin de génotyper les variations structurales découvertes par Conrad et coll. [48] chez

13. Utilisation du *Genome Analyser* d'*Illumina* [98].
14. Même nombre de copies chez l'individu de test et de référence.
15. Présence d'un nombre de copies différent de 2 chez *NA10851*.

un total de 19 050 individus (environ 2 000 individus pour chacune des huit maladies communes à l'étude ainsi qu'environ 3 000 individus témoins). En ajoutant l'information de plusieurs autres puces de SNP, un total de 11 541 régions ont été ciblées et plusieurs étapes de contrôle de qualité ont permis d'obtenir un total de 3 432 régions polymorphiques de bonne qualité (concordance de 99,7 % chez les individus dupliqués). La majorité de ces CNV (88 %) possédait deux ou trois classes (c.-à-d. bi-alléliques). Environ 44 % des CNV autosomaux possédaient une fréquence de l'allèle mineur de moins de 5 % (c.-à-d. rares). Ils ont ensuite vérifié le taux de déséquilibre de liaison entre les CNV et les SNP afin de déterminer si les GWAS précédents avaient indirectement exploré les CNV dans leurs analyses. Pour les régions de haute qualité avec une fréquence de plus de 10 %, 79 % d'entre elles possédaient un $r^2 > 0{,}8$ avec au moins un SNP. Pour celles ayant une fréquence de moins de 5 %, 22 % d'entre elles possédaient un $r^2 > 0{,}8$ avec au moins un SNP. Cette étude a démontré que les CNP communs ont donc pu être indirectement analysés par la majorité des GWAS précédents, n'ajoutant pas d'information supplémentaire au problème du manque d'héritabilité rencontré par les études génétiques. Il s'agit de l'une des premières études qui démontrent l'importance d'analyser de manières différentes les CNP communs et les CNV rares.

1.3.3.6. Séquençage à haut débit

En 2007, Korbel et coll. [8] ont utilisé la cartographie d'extrémités appariées avec des fragments d'une longueur de 3 kb. Le séquenceur 454 [100] de la compagnie *Roche* a été utilisé, faisant appel au pyroséquençage [101] [16]. Leur étude permettait de détecter les insertions, les délétions et les inversions de 3 kb et plus ainsi que les insertions simples de 2 à 3 kb chez deux individus sains (une femme de descendance européenne probable [*NA15510*] ainsi qu'une femme de descendance africaine (YRI) [*NA18505*]). La précision de la méthode était de 644 paires de bases. Il y a eu une détection d'un total de 1 297 événements de variations structurales, soit 853 délétions, 322 insertions et 122 inversions. Ces régions étaient distribuées à travers le génome. Un peu moins de la moitié (45 %) des variations structurales prédites étaient partagées par les deux individus. La majorité des variations détectées étaient petites (65 % < 10 kb et 30 % < 5 kb). Par contre,

16. Technique de séquençage basée sur la détection de pyro-phosphate (PPi) durant la synthèse de l'ADN. De la lumière visible est générée de manière proportionnelle au nombre de nucléotide incorporé.

certaines variations (15 %) étaient plus grandes que 100 kb. La distribution des longueurs des variations était semblable chez *NA15510* et *NA18505*. Une petite fraction des variations analysées (17 %) pouvait affecter directement la fonction de gènes en supprimant des exons ou en fusionnant des gènes (40 gènes), en étant localisée dans un intron (243 gènes) ou en altérant le nombre de copies d'un gène ou son orientation (32 gènes). Korbel et coll. furent les premiers à développer une méthode permettant la cartographie d'extrémités appariées afin de procéder à la détection des variations structurales, dont les CNP et les inversions.

Finalement, en 2011, Mills et coll. [102] ont utilisé les données du *1000 Genomes Project* afin de faire la détection de variations structurales chez 185 individus (59 Yoruba du Niger [YRI], 60 Européens du Utah [CEU], 30 Han de Beijing [CHB] et 30 Japonais de Tokyo [JPT] séquencés à faible couverture ainsi que deux trios YRI et CEU séquencés à haute couverture). Utilisant plusieurs méthodes de détection différentes (dont la profondeur de couverture et les extrémités appariées), ils ont détecté un total de 22 025 délétions et 501 duplications en tandem, 5 371 insertions d'éléments mobiles et 128 insertions absentes chez la référence. La longueur médiane des variations structurales était de 729 paires de bases avec une moyenne de 8 kb. Il s'agit ici d'une importante réduction de la longueur des variations trouvées, lorsque comparées avec Conrad et coll. [48]. Plusieurs variations (1 775) affectaient des régions codantes. Utilisant une technique permettant d'ajouter de l'information à propos des populations, ils ont réduit le nombre de délétions autosomales à 13 826 chez 156 individus. Ces génotypes présentaient une concordance de 99,1 % avec les données aCGH de Conrad et coll. [48]. Les délétions communes (fréquence de l'allèle mineur > 5 %) étaient partagées à travers les différentes populations. Ils ont observé une déplétion de délétions à haute fréquence qui chevauchait des gènes (bien que certaines délétions de régions codantes avaient une fréquence de plus de 80 %). Une grande majorité de délétions communes avait un déséquilibre de liaison $r^2 > 0{,}8$ avec au moins un SNP. Il s'agit d'une des premières recherches qui a permis l'étude des variations structurales avec une résolution nucléotidique en utilisant des données de séquençage à haut débit avec deux méthodes différentes (profondeur de couverture et extrémités appariées).

1.3.4. Progression des connaissances

Toutes les études présentées précédemment ont permis d'approfondir nos connaissances sur les variations en nombre de copies rares et communes. À mesure que les technologies de détection des CNV et des CNP se sont améliorées, une meilleure résolution a été acquise, permettant d'estimer correctement la taille de ces variations ainsi que leur mode de création. Il a aussi été montré que les CNV et les CNP constituaient effectivement une bonne partie de la variabilité interindividuelle même si aucune proportion précise n'a été calculée et que leur taille moyenne avait été préalablement surestimée. Aussi, il a été démontré que les CNP communs n'ont probablement pas beaucoup d'information à fournir dans le cadre d'études d'association avec des maladies complexes, à cause du fort déséquilibre de liaison avec les SNP avoisinants. Par contre, puisque les conclusions concernant les CNP communs ne sont pas applicables aux CNV rares, beaucoup de travail reste à faire afin de mieux comprendre leur implication dans la variabilité des traits complexes.

La majorité des CNV et des CNP découverts se retrouvent maintenant dans une base de données nommée *Database of Genomic Variants* (DGV) [9]. Depuis ses débuts en 2004, elle est passée d'environ mille à maintenant plus de 2,5 millions d'entrées provenant de 55 études distinctes sur plus de 22 300 génomes. La majorité des données incluses au début provenaient d'études avec des micropuces à faibles résolutions sur des nombres limités d'échantillons. Plusieurs de ces études ont maintenant été retirées de DGV, par leur haut taux de faux positif et de faux négatif. Des données provenant de micropuces à haute résolution et de séquençage à haut débit peuplent maintenant DGV, augmentant ainsi la précision du catalogue (voir Figure 1.10).

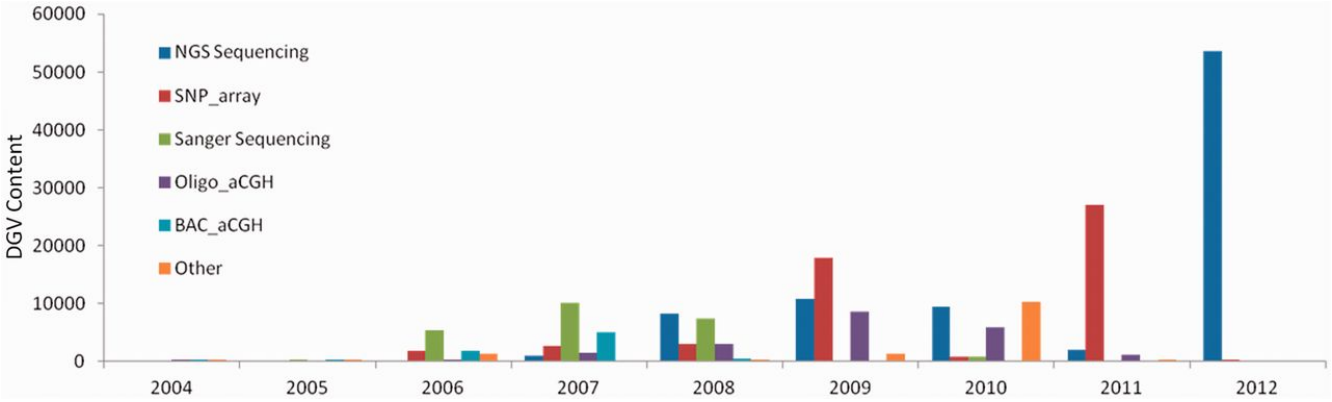


FIGURE 1.10. – Contenu de la base de données DGV (*Database of Genomic Variants*). Augmentation du nombre de variations reportées à DGV, montrant la transition récente vers l'utilisation de technologies possédant une meilleure résolution. Figure tirée de [9].

2. Analyses génétiques

Les analyses génétiques de maladies humaines peuvent être groupées en deux catégories générales : les études familiales de liaison génétique (*linkage*) et les analyses populationnelles d'association (à travers le génome ou pour des gènes candidats) [42]. Généralement, le type de désordres génétiques à l'étude dictera le choix du type d'analyse et des marqueurs génétiques utilisés dans le cadre d'un projet.

2.1. Les désordres génétiques

Les désordres génétiques sont des maladies causées par une ou plusieurs anomalies génomiques. Bien que certaines maladies peuvent être causées uniquement par la présence d'un ou de deux allèles mutés, des composantes environnementales ainsi qu'une interaction entre différents gènes peuvent augmenter le risque d'en être atteint [103]. Parmi les différents désordres causés entièrement ou partiellement par des facteurs génétiques, trois grandes catégories sont reconnues [104] : les désordres monogéniques, les syndromes chromosomiques et les désordres multifactoriels. Les désordres monogéniques (*single-gene disorders*) résultent d'une ou de plusieurs mutations sur l'un ou sur les deux allèles d'un gène situé sur un autosome, sur un chromosome sexuel ou sur de l'ADN mitochondrial [103]. Même s'ils sont rares (lorsque considéré individuellement), les désordres monogéniques sont responsables d'une proportion non négligeable de maladies et de décès (soit environ 2 % de la population) [104]. Les syndromes chromosomiques sont présents s'il y a une altération visible dans le nombre ou la structure des chromosomes [103]. Les différentes maladies sont donc causées par un excès ou un déficit des gènes contenu dans certains segments de chromosome, ou sur des chromosomes en entier. Les désordres multifactoriels découlent de l'interaction d'un gène ou plus avec un ou plusieurs facteurs environnementaux. La contribution génétique prédispose

donc l'individu concerné à l'action des différents agents environnementaux. Dans la plupart des cas de désordres multifactoriels, la nature des agents environnementaux ainsi que la prédisposition génétique ne sont pas connues et sont sujets de recherches intensives [103].

2.2. Analyses de liaison génétique

Les analyses de liaison génétique sont utilisées afin d'identifier des régions du génome qui contiennent des gènes prédisposant à une maladie transmise de façon héréditaire dans une ou plusieurs familles. Deux loci sont liés génétiquement s'ils sont transmis ensemble d'un parent à sa descendance plus souvent qu'attendu par le hasard [10]. Le but d'une étude de liaison génétique est donc d'identifier des marqueurs génétiques présentant une faible probabilité de recombinaison avec le locus du trait étudié, comparativement à l'hypothèse nulle de recombinaison aléatoire. Il existe deux types d'études de liaison génétique : (1) la liaison paramétrique (ou *model-based*), qui nécessite une connaissance *a priori* du mode et des paramètres de transmission génétique du trait, et (2) la liaison non paramétrique (ou *model-free*), plus souvent utilisée pour l'étude de traits complexes, puisqu'elle ne dépend pas de spécification adéquate d'un modèle de transmission [105]. Ce type d'étude de liaison implique généralement l'étude de fratries [1] et des allèles identiques par descente (*identical by descent* ou IBD) ou identiques par état (*identical by state* ou IBS) (Figure 2.1). Il est ainsi possible de positionner le locus d'une maladie en estimant le taux de recombinaison entre ce dernier et un marqueur (analyse deux points) ou un groupe de marqueurs simultanément (analyse multipoint), et ce, selon le positionnement génomique des marqueurs [106].

Bien que les études familiales de liaison génétique aient permis de découvrir des déterminants génétiques pouvant expliquer une susceptibilité ou même causer une maladie monogénique, elles ont eu moins de succès pour les maladies communes et multifactorielles. Ceci est dû au fait qu'il y a un manque flagrant de puissance, sauf lorsqu'en présence d'un simple locus pouvant expliquer une fraction substantielle de la maladie [42], ce qui n'est toutefois pas le cas pour les maladies complexes.

1. Frères et soeurs.

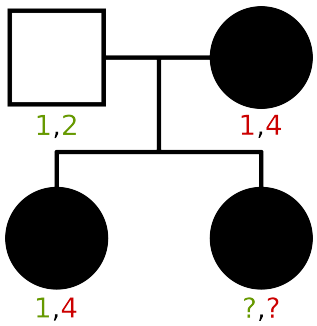
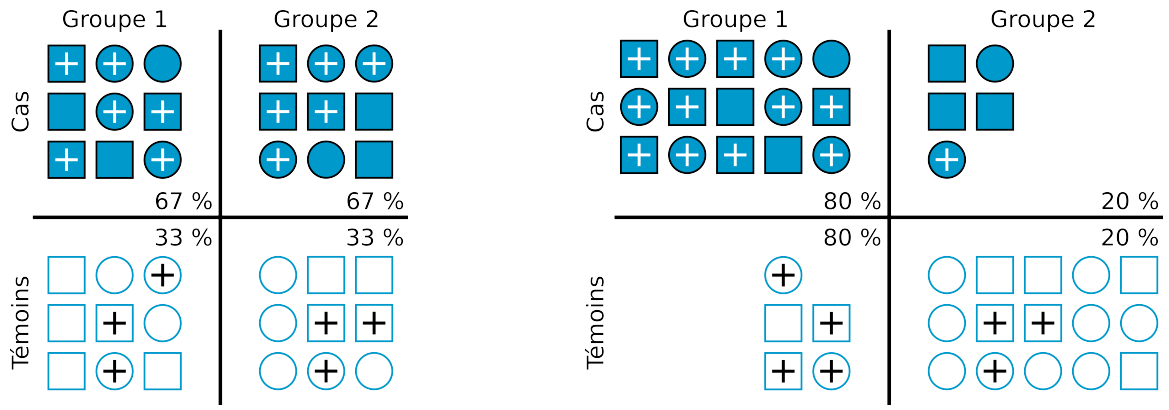


FIGURE 2.1. – Partage d'allèles entre fratries pour analyse de liaison génétique non paramétrique. Les allèles provenant du père sont en vert et ceux de la mère, en rouge. Il y a quatre génotypes possibles pour la deuxième soeur : (1,1), (1,4), (2,1) ou (2,4). Le nombre d'allèles IBD est 1, 2, 0 et 1, respectivement. Le nombre d'allèles IBS est 1, 2, 1 et 1, respectivement. Dans le cas du génotype (2,1), l'allèle 1 est le même (= IBS) mais ne provient pas du même parent (≠ IBD). Figure adaptée de Teare et coll. [10].

2.3. Analyses d'association génétique

Une étude d'association génétique est définie comme étant une analyse faisant usage de variations génétiques communes distribuées à travers le génome et visant l'identification d'associations avec un phénotype ou un trait à composante génétique [107]. Contrairement aux études de liaison génétique, les analyses d'association utilisent généralement des populations non reliées de cas (individus possédant le phénotype d'intérêt) et de témoins (ne le possédant pas). Le test d'association génétique se fait généralement un marqueur à la fois, à l'aide d'un test approprié, tel que le test de χ^2. Il suffit simplement de classer les individus selon leur statut (cas ou témoins) et selon le nombre d'allèles (mineur a et majeur A) ou le nombre de génotypes (a/a, a/A et A/A) dans une table. Sous l'hypothèse nulle de non-association avec le phénotype, il est attendu que la fréquence des allèles (*test d'association allélique*) ou des génotypes (*test d'association génotypique*) est la même chez les cas et chez les témoins. D'autres modèles alternatifs peuvent être utilisés. Par exemple, afin de tester un *modèle de dominance*, le nombre de génotypes A/A est comparé au nombre total de génotypes a/A et a/a combinés. Afin de tester un *modèle récessif*, le nombre de génotypes a/a est comparé au nombre total de génotypes a/A et A/A combinés. Le test d'association peut aussi être réalisé par des modèles de régression faisant appel à la méthode du rapport de vraisemblance (*likelihood ratio*). Ces modèles permettent l'ajout de co-variables additionnelles pouvant influencer les phénotypes



(a) Vraie association **(b)** Biais de confusion

FIGURE 2.2. – Biais de confusion causé par un mélange ethnique. Les formes pleines représentent les individus cas alors que les vides, les individus témoins. Le symbole + représente la présence du variant à l'étude chez l'individu. Lors d'une vraie association (2.2a), la fréquence de l'allèle de risque est plus grande chez les cas que chez les témoins (et ce, dans les deux groupes ethniques). En présence d'un biais de confusion (2.2b), la fréquence est la même entre les cas et les témoins d'un même groupe ethnique, mais apparaît globalement environ deux fois plus grande (65 % chez les cas et 35 % chez les témoins). Ceci est dû au fait que le groupe ethnique 1 est surreprésenté chez les cas. Figure adaptée de Hirschhorn et coll. [11].

complexes [108]. Ce type d'association génétique permet de trouver des polymorphismes affectant directement le phénotype (association directe) ou soit en déséquilibre de liaison avec la variation causale (association indirecte).

Les études d'association génétique sont sujettes à un possible biais de confusion, causé par une structure dans les données imputable à l'origine populationnelle des sujets. La présence d'une différence systématique dans les fréquences des allèles chez différentes sous-populations (Figure 2.2) en est un exemple. Si une de ces sous-populations possède une prévalence de la maladie plus forte, elle pourrait être surreprésentée dans les cas (Figure 2.2a), comparativement aux témoins (Figure 2.2b). Tout polymorphisme en plus grande abondance dans cette sous-population pourrait alors être faussement associé au phénotype dans cette étude (faux positifs) [11].

2.4. Problème de l'héritabilité manquante

Depuis 2008, plus de 14 000 études d'association génétique ont été réalisées sur plus de 800 maladies et autres phénotypes [109]. Toutes ces études se basent sur l'hypothèse que les maladies communes sont attribuables en grande partie à certains variants communs, présents chez plus de 1 à 5 % de la population [110]. D'ailleurs, les technologies permettant de déterminer le génotype d'une grande quantité de marqueurs chez plusieurs individus à la fois, les micropuces de SNP, ne proposent généralement qu'un échantillonnage de marqueurs relativement communs dans les populations du HapMap. Malheureusement, toutes ces études n'ont permis d'expliquer qu'une faible partie de l'héritabilité de la plupart des traits complexes [82]. Par exemple, plusieurs études ont démontré que l'héritabilité de la taille chez l'homme se situe entre 80 et 90 %. Ainsi, si 29 cm séparent les 5 % plus grands des plus petits dans une population donnée, l'élément génétique expliquerait jusqu'à 27 de ces centimètres. Malgré le fait que plusieurs marqueurs communs ont été associés à la taille d'une personne, ceux-ci n'avaient qu'un petit effet sur le phénotype, n'expliquant qu'environ 5 % de l'héritabilité, soit environ 6 cm [111]. Plus récemment, avec l'aide de larges études d'associations, la proportion de l'héritabilité de traits complexes expliquée par la composante génétique a augmenté jusqu'à 20 à 30 %, mais la majorité de l'héritabilité reste inexpliquée [112].

Cette héritabilité manquante pourrait être expliquée par différents facteurs, les plus importants étant les variations rares, les variations structurales, les interactions entre gènes, les interactions entre gènes et environnement, et plus particulièrement l'épigénétique [110, 111, 112, 113, 114]. L'épigénétique est définie comme étant des changements héréditaires causés par l'activation et la désactivation des gènes, sans altérations de la séquence d'ADN [115, 114]. Ces changements peuvent être engendrés par des stimuli environnementaux et comprennent plusieurs mécanismes comme la modification des histones (jouant un rôle sur l'état de la chromatine) et la méthylation des résidus cytosine de l'ADN de manière spécifique au tissu [115, 116].

La qualité des données analysées (données génétiques et phénotypiques) peut jouer un grand rôle sur les résultats d'un test d'association génétique. Les erreurs de génotypage, spécialement si elles

apparaissent différemment entre les cas et les témoins, sont une importante source de fausses associations. Elles doivent donc être trouvées et corrigées avant toute étude génétique [107, 117, 118]. Il est aussi recommandé de vérifier les graphiques de clustering, afin d'inspecter visuellement la qualité des génotypes des SNP jugés comme étant significativement associés au trait à l'étude pour s'assurer que cette association n'est simplement pas le reflet d'un artefact de génotypage [107, 119, 120].

Plusieurs arguments sont en faveur de l'hypothèse que les maladies complexes soient causées par un ensemble de variations nucléotidiques ou structurales rares [121]. Entre autres, la théorie de l'évolution prédit que les variations causant des maladies devraient être rares dans une population, car les maladies devraient diminuer l'aptitude à se reproduire [121, 122]. Aussi, des études empiriques ont démontré que les variations néfastes sont généralement rares [121]. De plus, la majorité des variations présentes dans l'exome humain le sont avec une faible fréquence (moins de 0,5 %) et sont spécifiques aux différentes populations étudiées [123, 124]. Finalement, la contribution des CNV rares à des désordres psychologiques complexes a déjà été prouvée [121]. Par contre, puisque plusieurs variations communes ont déjà été associées à des maladies complexes, il serait important de porter une attention accrue au degré d'implication des variations communes et rares à l'étiologie des maladies complexes. C'est pourquoi il est crucial d'employer des stratégies combinées permettant de découvrir à la fois ces deux types de variations [125]. Par exemple, bien que plusieurs résultats démontrent que les CNP communs ne pourraient apporter plus d'information aux études génétiques courantes [99], il est important de bien caractériser la séquence sous-jacente, car ces régions sont l'objet de mutations récurrentes de toutes sortes et il est difficile de prédire si un SNP avoisinant permettra de tous les analyser correctement [113].

3. Sujets abordés

L'objectif général de cette thèse vise le développement d'outils bio-informatiques et de nouvelles méthodologies permettant l'utilisation des SNP et des SNV de haute qualité ainsi que des CNP et des CNV dans le cadre d'études génétiques. Il est attendu que ce faisant, il sera possible d'expliquer une plus grande partie de l'héritabilité manquante des traits complexes, poussant ainsi l'avancement des connaissances sur les maladies chez l'homme.

Afin d'obtenir rapidement et en tout temps des données d'excellente qualité afin d'améliorer la performance des analyses génétiques et d'augmenter le taux de réplication des résultats obtenus, nous avons développé un outil facilitant et automatisant le nettoyage de données de SNP et de SNV (Chapitre 4). Cet outil automatise toutes les étapes du nettoyage des données, ce qui permet de réduire les risques d'erreurs et d'augmenter la puissance de détection. L'outil a été validé et grandement documenté (voir Annexe III à la page lxv) afin de permettre une installation et une utilisation faciles, efficaces et d'intérêt général. Nous en avons fait l'usage dans une étude clinique nécessitant une documentation analytique rigoureuse, et ce, avec grand succès [126]. Par la suite, afin de franchir les frontières de l'héritabilité manquante, notre laboratoire a fait l'usage d'une nouvelle micropuce de SNV, la *HumanExome BeadChip* (Illumina), pour faire le génotypage de plusieurs centaines de milliers de marqueurs rares chez un grand nombre d'individus. Par contre, étant donné la nouveauté de ce type d'outil, les algorithmes de clustering permettant de déterminer le génotype de chacun des individus pour chacun des marqueurs rares devaient être évalués. Nous avons donc fait une étude comparative de la qualité des algorithmes de clustering, permettant ainsi d'obtenir un jeu de données de meilleure qualité avec les variations rares (Chapitre 5).

Nous avons réalisé plusieurs projets permettant d'approfondir les connaissances sur l'implication des CNP et des CNV dans l'étiologie des traits complexes. Plus précisément, nous avons développé le premier outil bio-informatique permettant l'utilisation des CNP et des CNV pour les études familiales de liaison génétique (Chapitre 6). Pour ce faire, nous avons créé un algorithme permettant le partitionnement des CNP et des CNV selon le chromosome parental d'origine, permettant le phasage de ces marqueurs. Ceci est nécessaire aux études de liaison génétique. Ce partitionnement permet aussi la création d'haplotype afin de pousser plus loin les analyses avec ces marqueurs encore sous exploités. Nous avons aussi contribué à la réalisation d'expérimentations permettant de démontrer l'implication des CNV dans les maladies cardiaques congénitales du côté gauche du cœur [14]. Finalement, nous avons travaillé avec la collaboration du *Wellcome Trust Centre for Human Genetics* de l'*University of Oxford* à la caractérisation des défis liés à l'utilisation des CNV rares dans le cadre d'études d'association génétique (Chapitre 7). Pour ce faire, nous avons analysé les résultats générés par le *Wellcome Trust Case Control Consortium 2*, composés de données de micropuces d'ADN sur environ 9 000 individus sains. Après avoir généré la liste de tous les CNV rares provenant de ces individus, nous avons étudié l'impact des différents paramètres d'analyse sur les résultats d'association génétique.

Le développement d'outils et d'applications bio-informatiques pour les technologies émergentes est nécessaire à l'avancement des connaissances dans le domaine de la génomique et de l'épidémiologie génétique. Non seulement ces outils sont cruciaux afin de permettre la gestion efficace d'une grande quantité de données de bonne qualité, mais ils permettent aussi l'intégration de nouvelles connaissances aux tests statistiques déjà existants.

Deuxième partie

Contributions Scientifiques

4. pyGenClean: Efficient tool for genetic data clean up before association testing

Louis-Philippe Lemieux Perreault a participé au développement de la méthodologie de l'outil, a effectué son implémentation ainsi que celle des outils d'accompagnement, a participé à la validation des différents outils et a rédigé le manuscrit. Sylvie Provost a participé au développement de la méthodologie, a participé à la validation et à la rédaction du manuscrit. Marc-André Legault a participé à l'implémentation de l'outil, à sa validation et à la rédaction du manuscrit. Amina Barhdadi a contribué aux choix des analyses statistiques incluses dans l'outil ainsi qu'à la rédaction du manuscrit. Marie-Pierre Dubé a participé au design de l'outil, a supervisé le travail et a revisé le manuscrit. Tous les auteurs ont lu et ont approuvé le manuscrit final.

Bioinformatics 2013, **29**(13): 1704-1705 doi:10.1093/bioinformatics/btt261

pyGenClean: Efficient tool for genetic data clean up before association testing

Louis-Philippe Lemieux Perreault[1,2,*], Sylvie Provost[1], Marc-André Legault[2], Amina Barhdadi[1] and Marie-Pierre Dubé[1,2,].

[1]Beaulieu-Saucier Université de Montréal Pharmacogenomics Centre, Montreal Heart Institute Research Center, 5000 Bélanger Street, Montréal, Canada

[2]Université de Montréal, Faculty of Medicine, 2900 chemin de la tour, Montréal, Canada

Abstract

Summary: Genetic association studies making use of high throughput genotyping arrays need to process large amounts of data in the order of millions of markers per experiment. The first step of any analysis with genotyping arrays is typically the conduct of a thorough data clean up and quality control to remove poor quality genotypes and generate metrics to inform and select individuals for downstream statistical analysis. We have developed *pyGenClean*, a bioinformatics tool to facilitate and standardize the genetic data clean up pipeline with genotyping array data. In conjunction with a source batch-queuing system, the tool minimizes data manipulation errors, accelerates the completion of the data clean up process and provides informative plots and metrics to guide decision making for statistical analysis.

Availability and Implementation: *pyGenClean* is an open source Python 2.7 software and is freely available, along with documentation and examples, from http://www.statgen.org.

4.1. Introduction

Genome-wide association studies and similar designs typically rely on the use of massive amounts of genotype data covering the genome of thousands of study participants. Before proceeding to statistical analysis with genotyping array data, quality control (QC) and data clean up is usually performed to identify poorly performing samples and failed genotypes and to produce the analysis set according to set genetic ancestry criteria. This is particularly important for association tests which can be sensitive to even small sources of systematic or random errors. When joined with large sample size and a large number of genotyped markers, small errors can cause loss of statistical power and spurious associations. Hence, a thorough QC is required, even though it might become computationally intensive [127]. PLINK [128] is a tool widely used for quality assessment [129, 130, 127], providing multiple quality metrics about markers and samples that allows efficient data management. Even though PLINK is a complete toolset, it lacks automation and some steps require a considerable amount of manual tuning. *pyGenClean* automates the QC procedure using PLINK, while providing the user with multiple summarization files and visual aids for quick identification of quality issues. It also allows for the parallelization of steps for servers with a DRMAA-compliant distributed resource management system. The tool consists of multiple standalone scripts that are linked together via a main script and a configuration file, the latter facilitating user customization.

4.2. Methods

The QC pipeline was designed to respond to the needs of users for quality assessment tools of both the samples and their markers, historical process logs, speed, and parametrization. Different QC protocols have also been previously described in the literature [129, 130, 127]. Our QC pipeline is constructed from a set of script modules, which can be parameterized and ordered according to specific project needs. Figure 4.1 displays the recommended pipeline structure, designed to optimize the QC, and minimize sample loss. The procedures can be separated in two classes: marker and sample QC procedures.

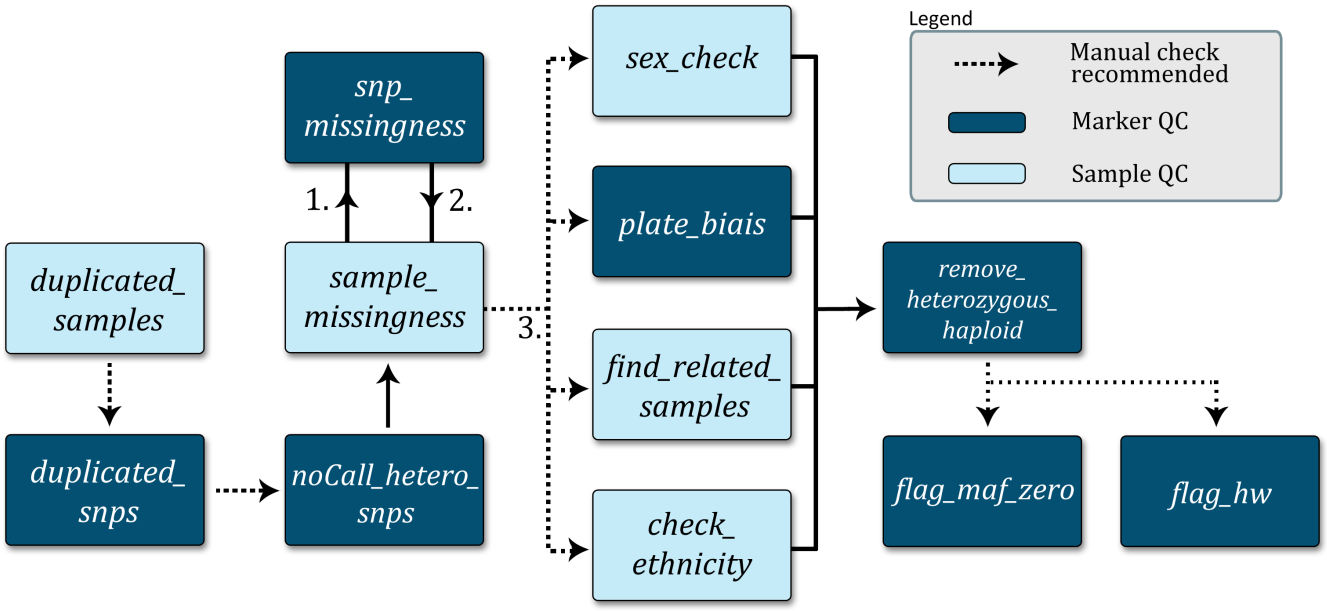


Figure 4.1.: Proposed data cleanup pipeline. Each box represents a customizable standalone script with a quick description of its function. Optional manual checks for go-no-go decisions are indicated. Numbers represent the ordering of the cyclic part of the pipeline.

The **marker QC** consists of seven scripts (Figure 4.1, in dark). The `duplicated_snps` module finds duplicated markers according to their chromosomal location (even if they have unique ids). It computes the completion and the overall concordance of each duplicates, and evaluates the possibility of merging the data of the duplicate markers when concordance criteria are met while zeroing out discordant genotypes. Since multi-allelic markers would appear to be duplicated in the dataset, the script will preserve incompatible duplicates. The `noCall_hetero_snps` module removes any marker with a heterozygosity rate (excluding the MT markers) or a missing rate of 100%. The `snp_missingness` module removes all the markers with a missing rate that is higher than the user-defined threshold. The `plate_bias` module removes marker showing significant plate bias as assessed by the comparison of allele frequencies from one plate to all others. The `remove_heterozygous_haploid` module zeroes out heterozygous haploid markers. The `flag_maf_zero` module flags markers with a minor allele frequency (MAF) of zero. The `flag_hw` module flags markers that fail Hardy Weinberg Equilibrium (HWE).

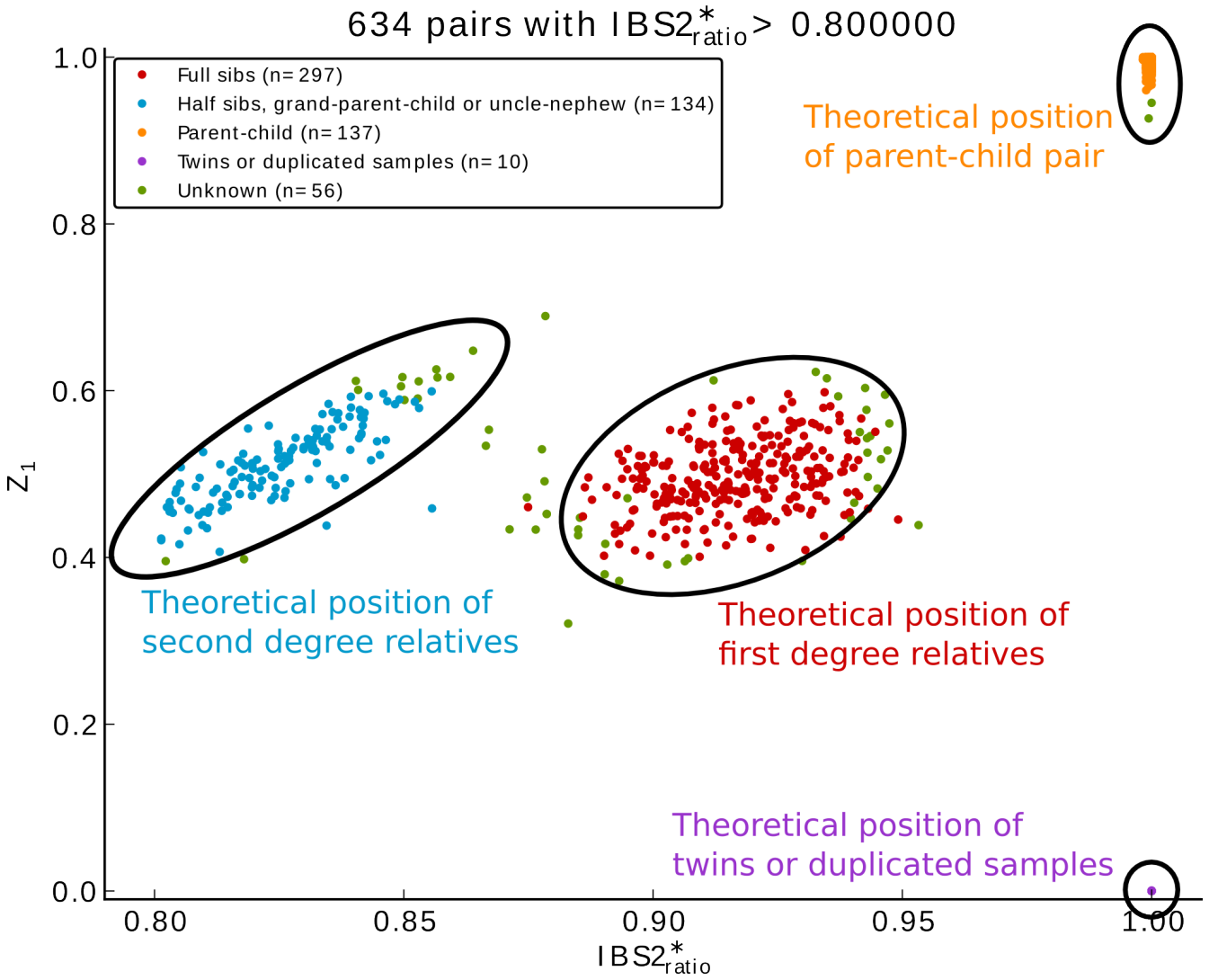


Figure 4.2.: Z_0 in function of $IBS2^*_{ratio}$ showing sample relatedness.

The **sample QC** steps consist of five scripts (Figure 4.1, in light). The `duplicated_samples` module helps in comparing duplicated samples with identical idenfitiers (ids). Since the PLINK program requires unique ids, input files for this script consist of transposed PLINK pedfiles (separated by tabulations) and the script performs the required computations. It computes the completion and the overall concordance of each duplicates, and evaluates the possibility of merging the data of the duplicate ids when concordance criteria are met and by zeroing out discordant genotypes. The `sample_missingness` module will remove all the samples that have a missing rate higher than a user-defined threshold. The `sex_check` module uses PLINK's check sex options to compare gender registered in the input files and the gender deduced by their X chromosome heterozygosity rate. Using all samples, the script also produces a graph showing the summarized Y intensities in function of the summarized X intensities of all samples, highlighting problematic samples [130, 127]. The script also produces a plot showing the overall Log R Ratio (LRR) and B Allele Frequency (BAF) for both X and Y chromosomes of problematic samples, helping in the identification of allelic imbalance [130] and in estimating the number of X and Y chromosomes. Finally, the script computes the heterozygosity rate on the X chromosome and the number of missing calls on the Y chromosome, helping in the resolution of possible gender mixups. The `find_related_samples` module uses PLINK's genome option to estimate the relatedness of study participants using identity-by-descent (IBD) and identity-by-state (IBS) for each sample pairs. It finds the possible degree of relatedness using the Cotterman coefficients as estimated by PLINK (Z_0, Z_1 and Z_2). The script also computes the $IBS2^*_{ratio}$ to produce two plots of Z_0 and Z_1 in function of $IBS2^*_{ratio}$ (above a user defined threshold) [131]. Those plots provide visual support for the identification of sample pair relatedness (see Figure 4.2). Finally, the script offers the possibility of randomly selecting one sample of each related group, excluding other related samples from the final dataset. The `check_ethnicity` module uses the PLINK program to compute the multidimensional scaling (MDS) values of the samples. This method require a pairwise IBD matrix to be computed which can be computationally demanding. Using parallelization, the *pyGenClean* script efficiently computes the MDS values using PLINK and, with the addition of reference populations (such as CEU, YRI and JPT-CHB), will find outliers with respect to a user-defined reference population and will create MDS graphs. The

outliers detection script uses the standard deviation of each cluster found by a K-Means algorithm.

Two additional scripts are provided for the automation of the QC pipeline. The first one, `subset`, helps to subset the dataset by excluding or selecting a set of markers or samples. The second one, `compare_gold_standard`, compares the current dataset to a gold standard. For example, if some samples from the 1000 Genomes Project were genotyped, the script compares the study genotypes to those of the reference 1000 Genomes Project data. If needed, markers are flipped according to their minor allele for comparability.

4.3. Application

A dataset comprising a total of 6,528 samples (including multiple duplicates of four HapMap samples and internal control samples) genotyped at the Beaulieu-Saucier Pharmacogenomics Centre on the Illumina HumanOmni2.5Exome BeadChip (2,567,845 markers including 42,822 duplicated markers) was processed with *pyGenClean*. A parameter file, as described in Figure 4.1, was created and used on a cluster with 10 nodes of 8 Intel® Xeon® CPUs @ 2.40GHz [with hyper-threading] and 47GB of random access memory for each node. The `sample_missingness` module was run two times (with 10% and 2% of missing calls thresholds) and the `snp_missingness` module was run in between to minimize data loss. To optimize computation speed, the `duplicated_samples` module was run independently of the others, and some script were run in parallel on the cluster. All other scripts were run with default parameters.

After 4 days of computation (including manual verification time) the final dataset after QC consisted of 2,059,052 markers genotyped on 5,749 unique samples. Note that the full power of the cluster is used only for the relatedness and the ethnic modules, as only a maximum of 4 processes were used in parallel for the other modules.

As genetic datasets are getting larger, efficient genetic data QC and clean up procedures are required. *pyGenClean* ensures quick, customizable and traceable results with datasets of any size.

***Funding*:** This work was supported by the Montreal Heart Institute Foundation, Genome Canada and Genome Quebec.

Conflict of Interest: none declared.

5. Comparison of genotype clustering tools with rare variants

Louis-Philippe Lemieux Perreault a participé au design expérimental, a optimisé l'outil *GenoSNP* en modifiant les paramètres dans le code source, a comparé les différents outils, a implémenté les différents tests statistiques et a écrit le manuscrit. Marc-Andé Legault a contribué aux analyses et à la rédaction du manuscrit. Amina Barhdadi a contribué aux analyses statistiques et à la rédaction du manuscrit. Sylvie Provost a contribué au design expérimental, aux analyses et à la rédaction du manuscrit. Valérie Normand a contribué à l'optimisation manuelle de *GenCall* et à la rédaction du manuscrit. Jean-Claude Tardif dirige la Cohorte de l'Institut de Cardiologie de Montréal et a dirigé le projet génomique avec la puce *HumanExome*. Marie-Pierre Dubé a supervisé le projet, a participé au design expérimental et à la coordination de l'étude, a supervisé le travail en laboratoire et a revisé le manuscrit. Tous les auteurs ont lu et ont approuvé le manuscrit final.

BMC Bioinformatics 2014, **15**:52; doi:10.1186/1471-2105-15-52

Comparison of genotype clustering tools with rare variants

Louis-Philippe Lemieux Perreault[*,1,2], Marc-André Legault[2], Amina Barhdadi[1], Sylvie Provost[1], Valérie Normand[1], Jean-Claude Tardif[1,2] and Marie-Pierre Dubé[*,1,2]

[1]Beaulieu-Saucier Université de Montréal Pharmacogenomics Center, Montreal Heart Institute Research Center, 5000 Bélanger Street, Montréal, Canada

[2]Université de Montréal, Faculty of Medicine, 2900 chemin de la tour, Montréal, Canada

Abstract

Background: Along with the improvement of high throughput sequencing technologies, the genetics community is showing marked interest for the *rare variants/common diseases* hypothesis. While sequencing can still be prohibitive for large studies, commercially available genotyping arrays targeting rare variants prove to be a reasonable alternative. A technical challenge of array based methods is the task of deriving genotype classes (homozygous or heterozygous) by clustering intensity data points. The performance of clustering tools for common polymorphisms is well established, while their performance when conducted with a large proportion of rare variants (where data points are sparse for genotypes containing the rare allele) is less known. We have compared the performance of four clustering tools (*GenCall*, *GenoSNP*, *optiCall* and *zCall*) for the genotyping of over 10,000 samples using the Illumina's HumanExome BeadChip, which includes 247,870 variants, 90% of which have a minor allele frequency below 5% in a population of European ancestry. Different reference parameters for *GenCall* and different initial parameters for *GenoSNP*

were tested. Genotyping accuracy was assessed using data from the *1000 Genomes Project* as a gold standard, and agreement between tools was measured.

Results: Concordance of *GenoSNP*'s calls with the gold standard was below expectations and was increased by changing the tool's initial parameters. While the four tools provided concordance with the gold standard above 99% for common alleles, some of them performed poorly for rare alleles. The reproducibility of genotype calls for each tool was assessed using experimental duplicates which provided concordance rates above 99%. The inter-tool agreement of genotype calls was high for approximately 95% of variants. Most tools yielded similar error rates (approximately 0.02), except for *zCall* which performed better with a 0.00164 mean error rate.

Conclusions: The *GenoSNP* clustering tool could not be run straight "out of the box" with the HumanExome BeadChip, as modification of hard coded parameters was necessary to achieve optimal performance. Overall, *GenCall* marginally outperformed the other tools for the HumanExome BeadChip. The use of experimental replicates provided a valuable quality control tool for genotyping projects with rare variants.

5.1. Background

More than 13,500 genome-wide association studies (GWAS) of complex human diseases have been performed since 2008 [109]. The majority of GWAS were conducted using common single nucleotide polymorphism (SNP) arrays targeting markers that were identified from the international HapMap project [42, 43, 44]. These studies are based on the assumption that common traits are driven by common low-penetrance polymorphisms with a frequency of more than one or five percent in the population [110]. A vast proportion of the heritability of complex traits remains unexplained [112]. However, advances in genomic technologies now allow for the search of rare variants of modest to intermediate penetrance [132].

SNP arrays offer the possibility of rapid genotyping of thousands of samples with highly reliable results at low cost. Several commercial arrays now include a large fraction of rare single nucleotide variants (SNV) discovered by high-throughput sequencing technologies. The latter, while still expen-

sive compared to SNP arrays, allows for the discovery of all variants, rare and common, located in the genome of sequenced individuals. The Illumina HumanExome BeadChip provides a compromise between genotyping SNP arrays and next generation sequencing by enabling the genotyping of rare SNVs in thousands of samples at relatively low cost. The HumanExome BeadChip is enriched for rare and low frequency coding variations previously identified from the sequenced exomes of approximately 12,000 individuals of diverse populations for variations seen in more than two individuals and in more than two sequencing projects [133]. Compared to other genotyping platforms targeting millions of markers, the proportion of rare variants (minor allele frequency $< 5\%$) included in the HumanExome BeadChip is considerably larger.

A recent review of clustering tools for widely used Illumina BeadChip arrays was presented by Ritchie *et al.* [134]. They reported that some tools performed marginally better than others for common and rare variants (lowest frequency around 0.05). The authors noted that methods borrowing information from other SNPs (*e.g. within-sample* information) to genotype rare variants could theoretically outperform reference-based methods, as these would suffer from the limited information available in the training sets for the homozygous and heterozygous clusters of rare alleles. Such *within-sample* methods are implemented in *GenoSNP* [135]. Some tools, such as *M3* [136] and *optiCall* [137], use a mixture of *between-* and *within-sample* approaches. Other tools, such as *GenCall* (available in the *GenomeStudio* software) [138], rely on a reference cluster file to cluster marker genotypes one at a time. The *zCall* tool exclusively genotypes markers that have been previously "missed" by a another tool, and was also recently described [139].

For this project, the performance of *GenCall*, *GenoSNP*, *optiCall* and *zCall* for clustering markers from the HumanExome BeadChip have been analysed and compared. With the growing interest of the community for studies with rare variants [110, 111, 112, 113], this "head to head" comparison will provide guidance for study design, tool selection and results interpretation.

5.2. Methods

5.2.1. Clustering tools

Four clustering tools were compared: *GenCall*, part of the *GenomeStudio* software version V2011.1 [138], *GenoSNP* version 1.3 [135], *optiCall* version 0.3.3 [137] and *zCall* version 3.2 [139]. All four tools differ with respect to their genotype calling method.

GenCall (GenomeStudio)

GenCall is Illumina's proprietary tool and is available through the *GenomeStudio* software. For a complete description of this tool, refer to Ritchie *et al.* [134]. In brief, this tool uses the normalized microarray intensities for both alleles (noted X and Y) to compute the associated polar coordinates (r and θ) for each marker/sample pair. Next, using a *between-sample* model, meaning that it calls one marker by looking at the population of samples, it assigns genotypes by determining the nearest cluster using a reference containing the expected position of each genotype cluster for every marker as determined from the HapMap data. If required, the user may modify the position of each of the expected clusters to be more representative of the data at hand. By pre-assigning the expected position of each cluster, this method can readily provide a genotyping assignation of rare variants for studies having only a small number of samples. However, due to experimental variabilities and genomic variations in different populations, the position of the observed cluster's centroid might shift when compared with the expected one. A considerable amount of manual cluster adjustments might be needed to achieve good genotype calls.

For this project, a custom cluster file was created by modifying the expected position of the genotype cluster's centroid for a subset of markers by using all samples from the dataset. The markers selected for manual inspection were: (1) markers with a high heterozygous frequency, (2) markers with a low mean intensity, (3) markers with a low call frequency, (4) markers with a low minor allele frequency with no heterozygous calls, (5) markers showing an excess of heterozygous calls, (6) markers with

low AA T means or low BB T means [1], (7) mitochondrial markers, (8) markers on sex chromosomes, (9) markers that fail reproducibility tests, (10) markers with a small cluster separation or (11) markers with low quality score. To compare the efficiency of this modified cluster file, the results from *GenCall* with the original cluster file were also included in the analysis.

GenoSNP

GenoSNP uses a *within-sample* model, meaning that it assigns genotypes to all markers of a single sample at once [135]. It uses raw X and Y allele intensities extracted using the *GenomeStudio* software and calls genotypes by fitting a four-components mixture of Student's t-distributions on different subsets of markers (separated by Bead Pools) by the mean of a Variational Bayes Expectation Maximization algorithm (VB-EM). The use of this method improves robustness by allowing uncertainty in the statistical model, in contrast to standard expectation maximisation methods. *GenoSNP* computes the posterior probability of the marker genotype calls. As this tool calls one sample at a time, it offers the flexibility of providing final sample genotyping results before the whole study dataset is ready to be processed. It can also be parallelized by running the tool on multiple samples at a time, and it does not require a reference panel. It is generally expected that this tool would perform well with rare variants, as their genotypes will be clustered with higher frequency variants according to measured X and Y allele intensities (as opposed to *between-sample* methods, where rare variants are sparsely located in the heterozygous cluster).

To speed up the genotyping process, samples were called using *GenoSNP* as soon as they were released from the genotyping center. A posterior probability cutoff of 0.8 was used to achieve higher quality calls. To ascertain the quality of the results once all samples were genotyped, the mean and the median intensities of all calls for each sample were plotted.

optiCall

The *optiCall* tool uses a mixture of *between-* and *within-sample* models. It uses a subset of (X, Y) intensities from random samples at a random marker to find a prior distribution to the statistical

1. θ values of the center of the AA and BB clusters in normalized polar coordinates, respectively.

model used to call genotypes across markers. This distribution is inferred by using an EM algorithm to fit a four-class mixture of Student's t-distributions. The initial values used by the EM algorithm are obtained by using the *kmeans++* algorithm [140] and the individual genotype's *a priori* probabilities are assumed to be uniform (0.25 for every cluster, including the outlier's cluster). Then, a second mixture of t-distributions is used for the *between-sample* clustering, where the previously estimated *priors* are used in a Maximum-A-Posteriori (MAP) estimate of its parameters.

To measure the quality of genotype calls, *optiCall* relies on deviation from the Hardy-Weinberg Equilibrium (HWE). The tool will try to improve the genotype calls when the HWE test fails ($p < 5 \times 10^{-15}$) by fitting the previously described model without a prior.

zCall

This tool functions as a post-processing tool (after a default one has been used) [139]. The *zCall* tool separates the clusters for rare variants by partitioning the $(X,\ Y)$ intensity space using horizontal and vertical thresholds. Their positions are derived from the mean and variance of the homozygote clusters for common variants that were previously called and are scaled according to a *z-score* factor to optimize concordance with the default tool. Genotypes are then assigned with respect to their position relative to the *z-score* scaled coordinates. Accordingly, rare variants are called by defining a distance threshold. The homozygote threshold for the major allele is estimated from the first calling tool's genotypes, and the rare allele's threshold is estimated by linear regression from the means and standard deviations of X and Y intensities of common markers.

As recommended by the authors, *zCall* was used as a post-processing step after *GenCall* (*GenomeStudio*). Version 3.2 was used, where all z thresholds were derived from *GenomeStudio*'s final report, from which samples were filtered out based on call rate and global heterozygosity. A z threshold of 8 was used after comparing the concordance with the original calls (maximum of 99.27%). Only missing genotypes from the original *GenomeStudio* report were recalled by *zCall*.

5.2.2. Dataset

The four tools were applied to a dataset consisting of 10,517 unique samples from the Montreal Heart Institute (MHI) Cohort. We also included 95 experimental replicates of *NA17281*, 15 replicates of *NA17251* and 3 replicates of *NA12763* from the Coriell Institute (the latter being sequenced by the *1000 Genomes Project* [19]). Finally, 93 and 40 MHI cohort samples were replicated 3 and 2 times, respectively. All 10,520 samples (10,856 including replicated ones) were genotyped using the Illumina HumanExome BeadChip, assessing 247,870 markers including 140 insertions/deletions (which were discarded from the present analysis). Some of these markers (214,599) are present in NHLBI GO Exome Sequencing Project (ESP) database [141], and 93% of these are rare variants with a minor allele frequency (MAF) below 5% in the European American population according to the ESP database [141]. The research protocol was approved by the Montreal Heart Institute research ethics review board and all participants signed an informed consent.

5.2.3. Agreement between tools

We used Cohen's kappa (κ) and Fleiss' pi (π), two widely used statistics, to compute the extent of agreement between raters [142], or in this case, genotype calling tools. Cohen's κ computes the extent of agreement between two tools by first computing the overall agreement probability (Equation 5.1), using a two-way contingency table (Additional file 1: Table S1), for the distribution of n samples by tools (rater) and genotype category, where n_{kl} indicates the number of samples that tool 1 and 2 classified into genotypes k and l, respectively.

$$p_a = \sum_{k=1}^{q} \frac{n_{kk}}{n} \tag{5.1}$$

In a comparable contingency table, Cohen's κ is estimated using Equation 5.2.

$$\gamma_\kappa = \frac{p_a - p_{e|\kappa}}{1 - p_{e|\kappa}}, \text{ where } p_{e|\kappa} = \sum_{k=1}^{q} \frac{n_{Ak} n_{Bk}}{n^2} \tag{5.2}$$

where p_a is the observed proportion of agreement (Equation 5.1) and $p_{e|\kappa}$ is the proportion of agreement expected by chance.

The data can be summarized in a frequency table (Additional file 1: Table S2), where, for a given sample i and genotype k, r_{ik} represents the total number of tools that called genotype k for sample i. Fleiss' π is then defined by Equation 5.3 [143].

$$\hat{\gamma}_\pi = \frac{p_a - p_{e|\pi}}{1 - p_{e|\pi}} \begin{cases} p_a &= \frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{q}\frac{r_{ik}(r_{ik}-1)}{r(r-1)} \\ p_{e|\pi} &= \sum_{k=1}^{q}\hat{\pi}_k^2, \text{ and } \hat{\pi}_k = \frac{1}{n}\sum_{i=1}^{n}\frac{r_{ik}}{r} \end{cases} \tag{5.3}$$

The possible set of genotypes included the *no call* genotype, as all tools might agree that a marker is impossible to be categorized in either of the three genotype clusters (homozygous or heterozygous) due to quality issues (*e.g.* low intensity).

5.2.4. Error rates

Several methods for error rate estimation (ϵ) with pedigree data have been proposed and reviewed by Liu *et al.* for their use with unrelated samples [12]. The genotypic model (as defined in Equation 5.4) provides a proper estimation of the error rate and proposes different constraints on the parameter space in order to make the model parameters identifiable.

$$\epsilon = \frac{2(3C_1 + 3C_3 - 1) \pm \sqrt{[6(C_1 + C_3) - 4]^2 + 12(C_1 - C_3)^2}}{6} \begin{cases} C_1 &\simeq \frac{n_{AA}}{n} \\ C_2 &\simeq \frac{n_{AB}}{n} \\ C_3 &\simeq \frac{n_{BB}}{n} \end{cases} \tag{5.4}$$

The genotypic model was tested by Liu *et al.* for common variants. However, we found that the possible values of ϵ were out of bounds (*i.e.* negative or above one) with the HumanExome BeadChip data where a majority of markers are rare. This can be explained by the proportion of the minor allele in the population, p_1, which is almost null. For these cases, ϵ was approximated using $\epsilon \simeq (C_1 - C_3 + 1)/3$ (Additional file 1: Equation S1).

5.3. Results and Discussion

5.3.1. Clustering quality

GenoSNP

The *GenoSNP* tool returns the probability of belonging to one of the three genotype clusters (homozygous A, B or heterozygous AB) for each evaluated genotype. The maximal probability is used to define the genotype to be called. We observed a majority of samples with a high proportion of low quality calls (close to the 0.8 quality threshold used). Cluster plots were created for some of the poorly performing samples which raised concern about the calling quality (Figure 5.1A and 5.1D). Suspecting a lack of convergence, modifications to the tool were made to increase the number of iterations of the VB-EM procedure. This improved the quality of the clustering for some of the samples (Figure 5.1B and 5.1E). Further modifications to the initial X and Y intensities' variance parameter from 0.1 to 100 (in addition to the increased number of iterations) greatly improved the quality of the clustering tool (Figure 5.1C and 5.1F). To efficiently ascertain the effect of the optimized parameters on the calling quality, plots of the mean and the median of the maximal probability of each sample have been created (Figure 5.2A and 5.2B respectively), showing a net increase of the probabilities. To further improve the comparison between the original *GenoSNP* and the modified version (300 iterations and an initial X and Y intensities' variance of 100), the two tools (original and optimized) were used for comparison with the other tools in this study.

zCall

According to Goldstein *et al.* [139], the optimal value of the z threshold should be determined by trying different values of z to find the one with the most concordance to the original *GenCall* calls. Here, the optimal value of z was determined to be 8, having a concordance of 99.27% with the original data.

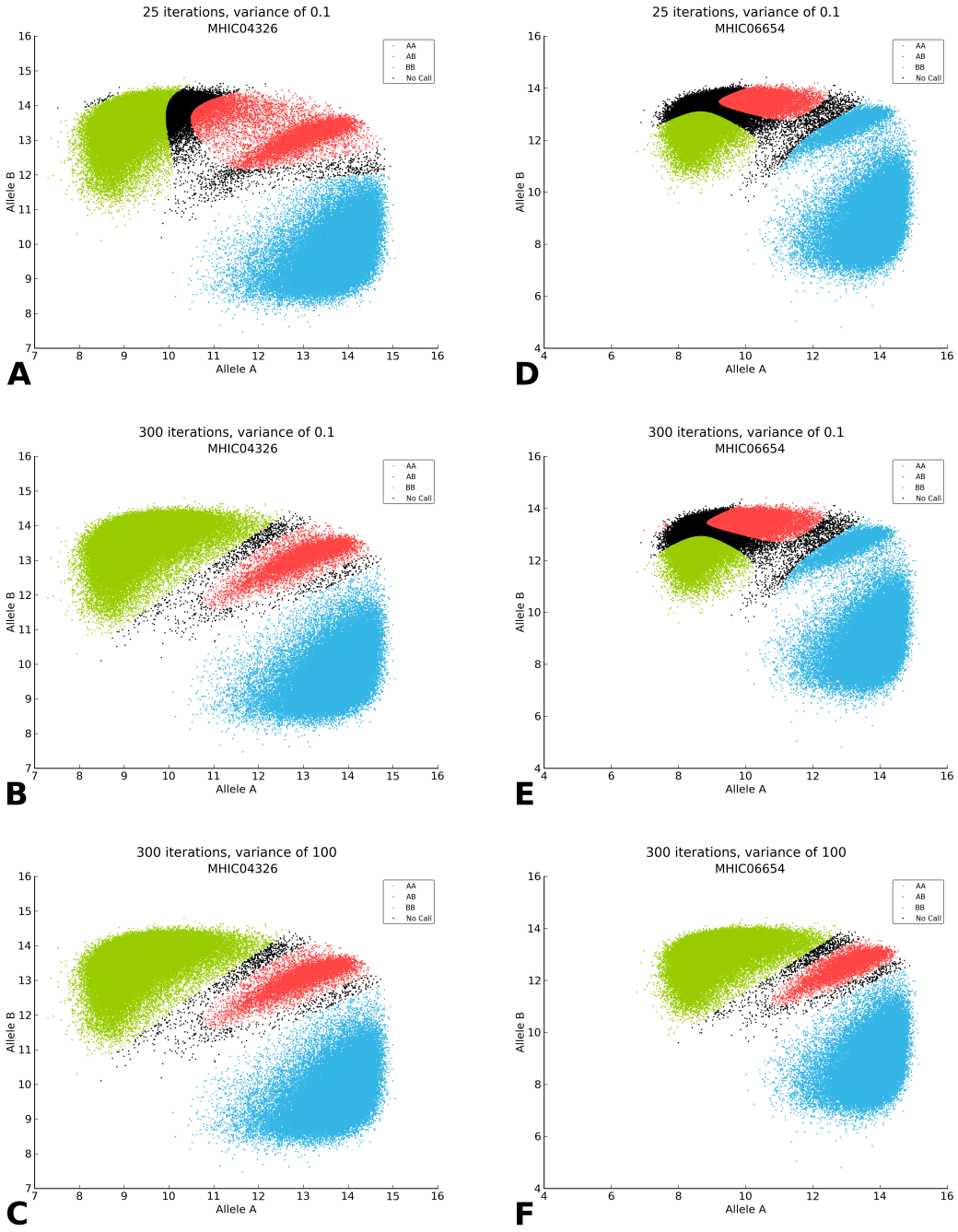


Figure 5.1.: Impact of parameters on GenoSNP's clustering. Impact of the choice of initial parameters on *GenoSNP*'s clustering of two samples (*MHIC04326* shown in (A), (B) and (C), and *MHIC06654* shown in (D), (E) and (F)). The first row shows the clustering results of the two samples using the default parameters (25 EM iterations and an initial variance of 0.1). The second row shows the results when keeping an initial variance of 0.1, but increasing the number of EM iterations to 300. The last row shows the results when increasing both the initial variance and the number of EM iterations to 100 and 300, respectively. In all cases, only markers in the BeadPool 1 are kept. Each point represents the raw B allele intensities against the raw A allele intensities (base two logarithm in both cases) of each marker. The *AA*, *AB* and *BB* genotypes are shown in blue, red and green, respectively. Markers that were below the quality threshold of 0.8 are shown in black.

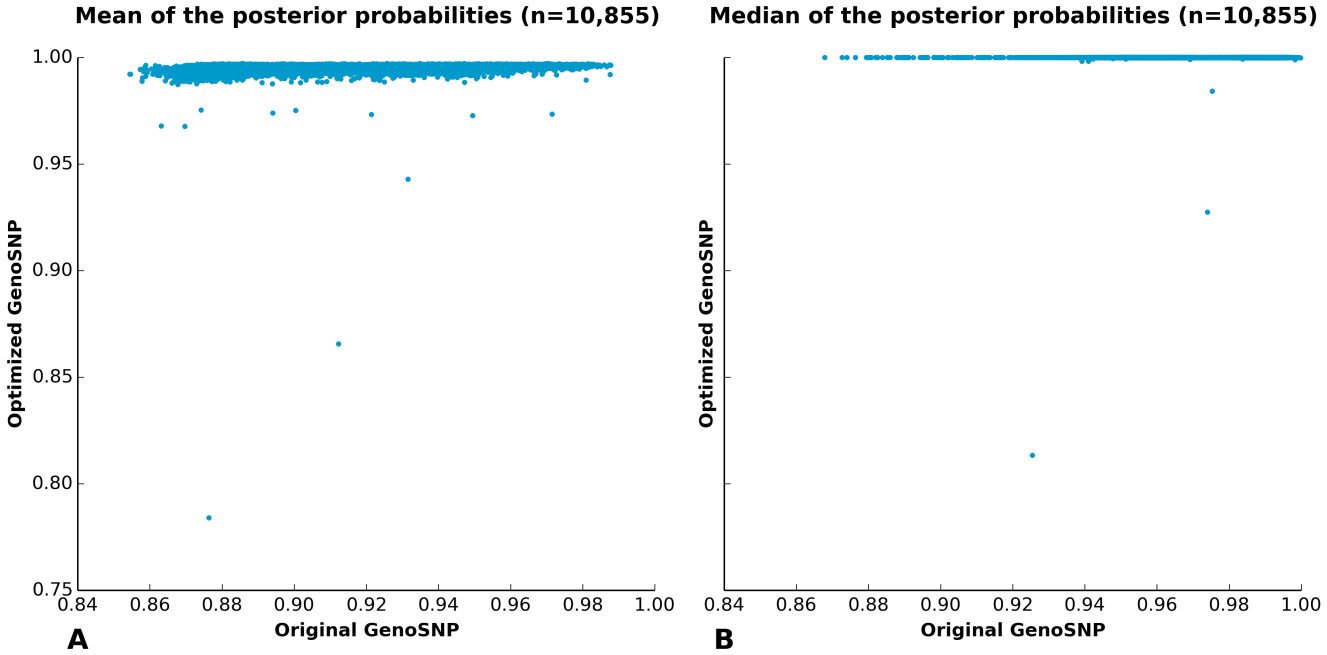


Figure 5.2.: GenoSNP clustering quality. Quality assessment of *GenoSNP*'s clustering for original (x axis) against optimized (y axis) parameters. For each sample, both the (A) mean and the (B) median of the posterior probability of all markers are shown. Each point represents a sample. A good clustering has a mean and a median calling probability close to one.

5.3.2. Missing rates

One important property of a calling tool is its capacity to assign a genotype to the majority of samples and markers (*i.e.* the calling rate, or inversely, the missing rate). The sample and marker missing rates of the six tools were compared (Figure 5.3A and 5.3B respectively). The original version of *GenoSNP* had the highest missing rate (19% for both sample and marker). The optimized version of *GenoSNP* increased both the mean sample and marker calling rates by 18.5% (from 80.6% to 99.1%). These rates were inferior to those of the other tools: 99.6%, 99.8%, 99.6% and 99.9% for *GenCall* (original and optimized cluster file), *optiCall* and *zCall*, respectively. It is important to note that the missing rate of *zCall* is bound to be less than or equal to that of *GenCall*, as *zCall* will only call missing genotypes from the results produced by *GenCall*. Also, the missing rates of *GenCall* were slightly better when using the optimized cluster file when compared to the original one. A non-parametric *Friedman Rank Sum* test comparing tools showed a statistically significant difference in both sample and marker missing rates (p-value $< 2.2 \times 10^{-16}$ in both cases),

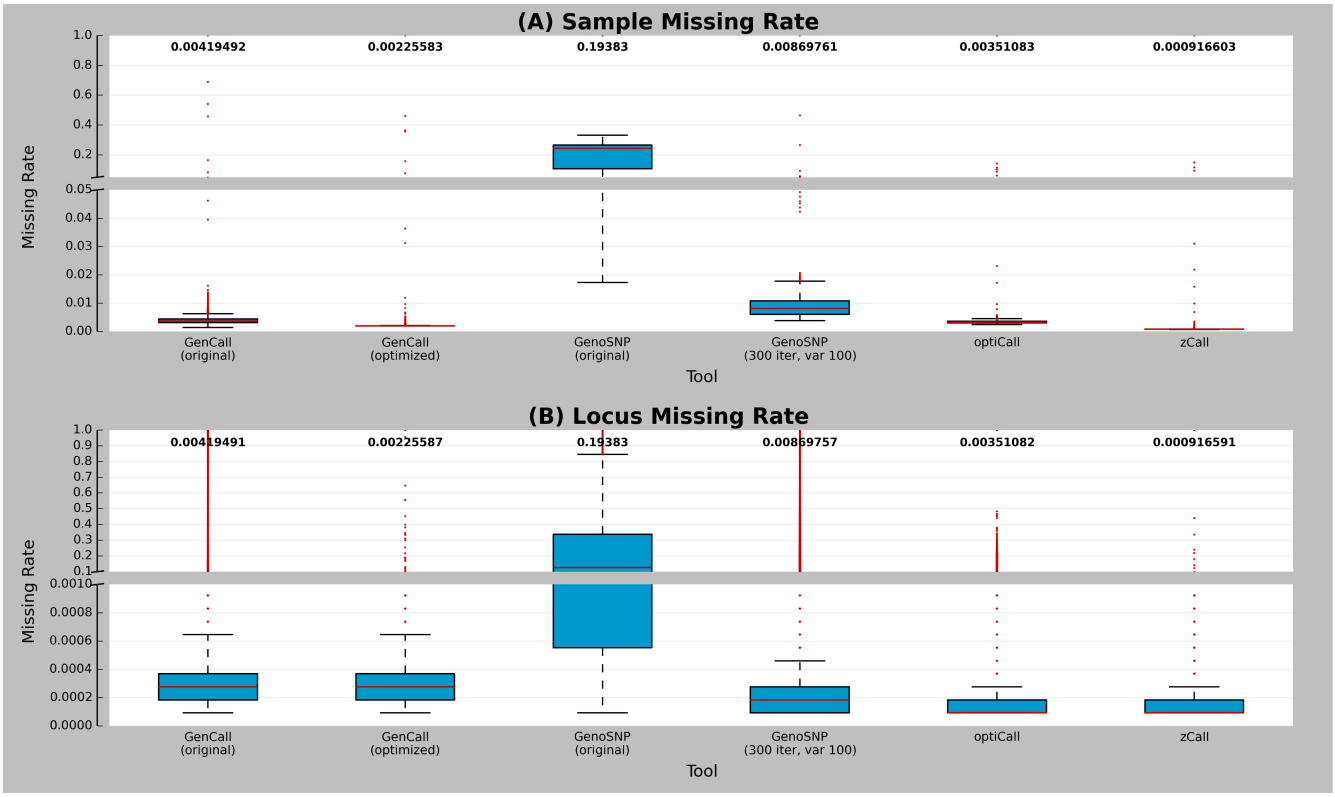


Figure 5.3.: Locus and sample missing rate. Boxplots showing the distribution of (A) locus and (B) sample missing rates. Values closer to 0 (100% completeness) are preferable. Rates are shown for each of the six tools: *GenCall* (original and optimized cluster files), *GenoSNP* (original and optimized), *optiCall* (without excluding markers failing Hardy-Weinberg) and *zCall*. The median is shown with a red line and the mean is indicated above each plot. The red dots represent outliers according to the interquartile range. The rates are shown for 10,856 samples and 247,590 markers (markers located on chromosome Y were excluded).

even though the rates seemed similar (Figure 5.3). Pairwise dependent-samples non-parametric sign tests comparing pairwise tools were also significant (p-value $< 2.2 \times 10^{-16}$ in all cases). More specifically, 0.55- and 0.83-fold decreases, on average, were observed for sample and marker missing rates, respectively, when comparing the original and the optimized cluster file for *GenCall*.

5.3.3. Precision estimates

The dataset contained a high number of technical replicates. The concordance of genotype calls between replicates was computed for the five tools (Figure 5.4). The optimized version of *GenoSNP* increased concordance between sample duplicates from a mean of 95.361% to 99.912%, but did not exceed the performance of *GenCall* (means of 99.996% for both the original and the optimized cluster

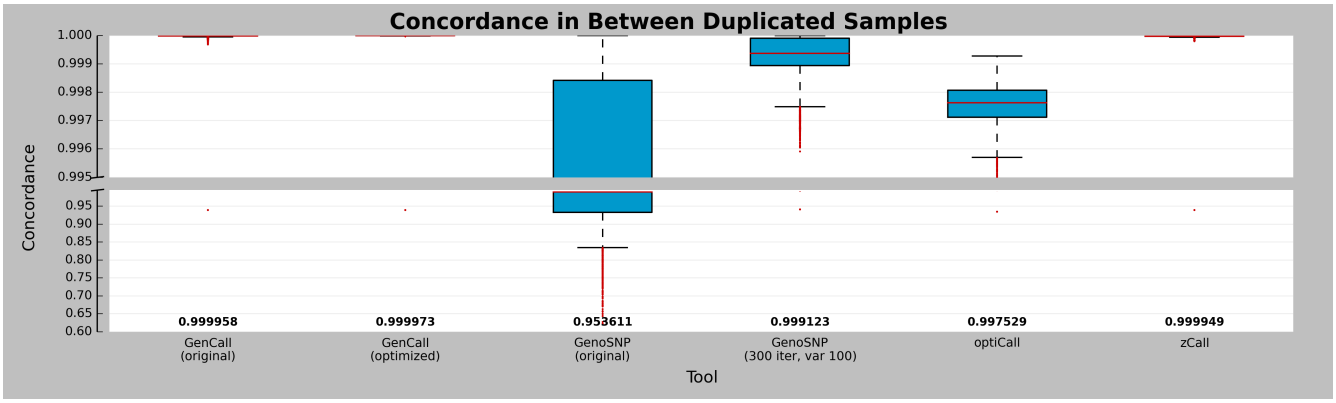


Figure 5.4.: Duplicated sample concordance. Boxplots showing the distribution of pairwise concordance of duplicated samples. Red lines are the median and the numbers at the bottom of the plot represent the concordance means. The red dots represent the outliers according to the interquartile range. The six tools are shown (from left to right): *GenCall* (original and optimized cluster files), *GenoSNP* (original and optimized), *optiCall* (without excluding markers failing Hardy-Weinberg) and *zCall*. A total of 4,892 sample pairs is shown.

files). Even though *zCall* reduced the missing rate from a mean of 0.4% to 0.09%, it added variability in the newly called genotypes, slightly decreasing the concordance between replicated samples when compared with the original results (*GenCall*, original cluster file). Overall, the call concordance between replicates was similar ($> 99\%$ concordance). *GenCall* had the highest concordance rate with a mean of 99.997% between replicated sample pairs. Optimizing *GenCall*'s cluster file had only a minor effect on the mean concordance, with a difference of $1.5 \times 10^{-3}\%$.

5.3.4. Accuracy estimates

One sample sequenced by the *1000 Genomes Project* was included three times to the original dataset to assess the concordance with next-generation sequencing. The genotypes called from the different tools were compared to the third release of the *1000 Genomes Project*, comprised of low coverage whole genome and high coverage exome sequencing data. The comparison with this gold standard was performed using the *pyGenClean* tool [144]. Since the majority of markers in the HumanExome BeadChip are located in exons, the results were mostly compared to the high coverage sequencing data (81.6% of the overlapping markers). Table v.1 shows the concordance for the three replicates of *NA12763*. Apart from the original *GenoSNP*, all the tools had concordance rates greater than 99%. *GenCall* (optimized cluster file) had the highest concordance rate (mean

Table V.I.: Call concordance with the *1000 Genomes Project* (all calls). Call concordance and number of compared markers for the three control replicates when compared to the *1000 Genomes Project*. The following six tools were compared: *GenCall* (original and optimized cluster files), *GenoSNP* (original and optimized), *optiCall* (without excluding markers failing Hardy-Weinberg) and *zCall*.

Tool	NA12763_R		NA12763_R1		NA12763_R2	
	Rate	Number	Rate	Number	Rate	Number
GenCall (original)	0.998591	127,062	0.998623	127,093	0.998425	126,980
GenCall (optimized)	0.998963	127,323	0.998963	127,323	0.998979	127,312
GenoSNP (original)	0.966771	113,365	0.907042	97,345	0.762193	93,555
GenoSNP (optimized)	0.996161	126,848	0.996446	126,893	0.995795	126,748
optiCall	0.995421	127,334	0.995413	127,323	0.994995	127,277
zCall	0.998785	127,545	0.998793	127,544	0.998785	127,545

of 99.897%), closely followed by *zCall* (mean of 99.879%). *GenCall* (original cluster file) also performed well, with a mean concordance of 99.855%. Then, calls were categorized according to their allele content: either (1) homozygous calls for the common allele or (2) involving the rare allele (heterozygous or homozygous calls for the rare allele). The frequencies were computed using the corresponding dataset (*i.e.* tool). Table V.II shows the concordance for the three replicates of *NA12763* for genotypes called as homozygous for the common allele. Apart from the original *GenoSNP* on one replicate, all had concordance rates greater than 99%. *GenCall* (original cluster file) had the highest concordance rate (mean of 99.948%), closely followed by *GenCall* (optimized cluster file) and *zCall* (mean of 99.939% and 99.928% respectively). Table V.III summarizes the concordance rates with the gold standard for genotypes involving the rare allele. *GenCall* (optimized cluster file) had the highest concordance rate (mean of 99.493%), closely followed by *zCall* (mean of 99.405%). The other tool had a lower concordance rate (*i.e.* between 95% and 97%).

5.3.5. Inter-tool agreement

To estimate tool agreement, three coefficients were computed: Cohen's κ and percent agreement (both shown in Figure 5.5), and Fleiss' π (Figure 5.6). Since *GenCall* (using the optimized version of the cluster file) provided the best result set, its agreement with the other three tools (excluding the original version of *GenoSNP*) was considered (Figure 5.5A and B). The best agreement was between *GenCall* and *zCall*, as expected due to the dependence of *zCall* on the results of *GenCall*.

Table V.II.: Call concordance with the *1000 Genomes Project* (homozygous common allele). Call concordance and number of compared markers for the three control replicates when compared to the *1000 Genomes Project*. For each dataset (*i.e.* tool), only genotypes called as homozygous of the common allele (according to the allele frequency computed using the corresponding dataset) were kept for analysis. The following six tools were compared: *GenCall* (original and optimized cluster files), *GenoSNP* (original and optimized), *optiCall* (without excluding markers failing Hardy-Weinberg) and *zCall*.

Tool	NA12763_R		NA12763_R1		NA12763_R2	
	Rate	Number	Rate	Number	Rate	Number
GenCall (original)	0.999478	114,975	0.999461	115,008	0.999486	114,876
GenCall (optimized)	0.999393	115,288	0.999384	115,288	0.999393	115,279
GenoSNP (original)	0.999630	97,236	0.998973	78,868	0.925482	74,210
GenoSNP (optimized)	0.999190	114,814	0.999138	114,910	0.999207	114,687
optiCall	0.998990	114,815	0.998972	114,805	0.999006	114,701
zCall	0.999281	115,438	0.999281	115,437	0.999290	115,438

Table V.III.: Call concordance with the *1000 Genomes Project* (heterozygous and homozygous rare allele). Call concordance and number of compared markers for the three control replicates when compared to the *1000 Genomes Project*. For each dataset (*i.e.* tool), only genotypes called as heterozygous or homozygous of the rare allele (according to the allele frequency computed using the corresponding dataset) were kept for analysis. The following six tools were compared: *GenCall* (original and optimized cluster files), *GenoSNP* (original and optimized), *optiCall* (without excluding markers failing Hardy-Weinberg) and *zCall*.

Tool	NA12763_R		NA12763_R1		NA12763_R2	
	Rate	Number	Rate	Number	Rate	Number
GenCall (original)	0.990155	12,087	0.990650	12,085	0.988351	12,104
GenCall (optimized)	0.994848	12,035	0.994931	12,035	0.995014	12,033
GenoSNP (original)	0.767847	16,179	0.514147	18,590	0.137233	19,507
GenoSNP (optimized)	0.967032	12,042	0.970397	11,992	0.963052	12,071
optiCall	0.962401	12,527	0.962478	12,526	0.958045	12,585
zCall	0.994053	12,107	0.994136	12,107	0.993970	12,107

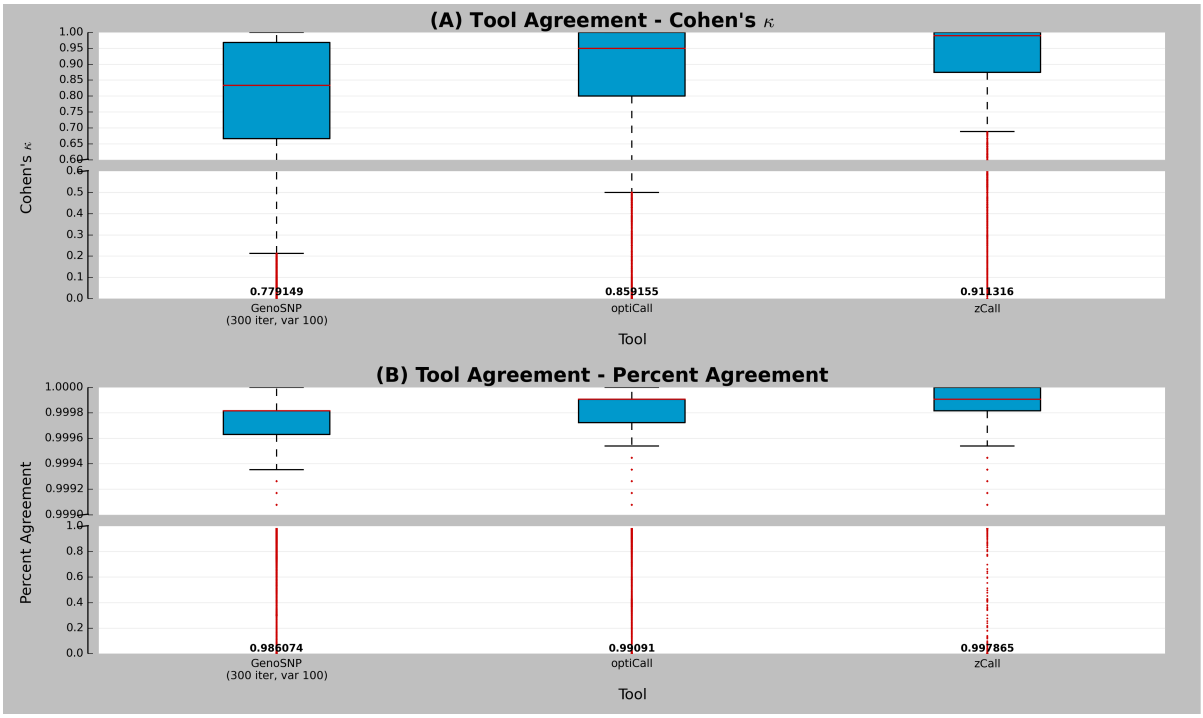


Figure 5.5.: Pairwise tool agreement. Boxplots showing the Cohen's κ coefficient (A) and the percent agreement (B) for all calls when *GenCall* is compared with the following three tools: *GenoSNP* (optimized), *optiCall* (without excluding markers failing Hardy-Weinberg) and *zCall*. Red lines are median and the numbers at the bottom of the plot represent the concordance means. The red dots represent the outliers according to the interquartile range.

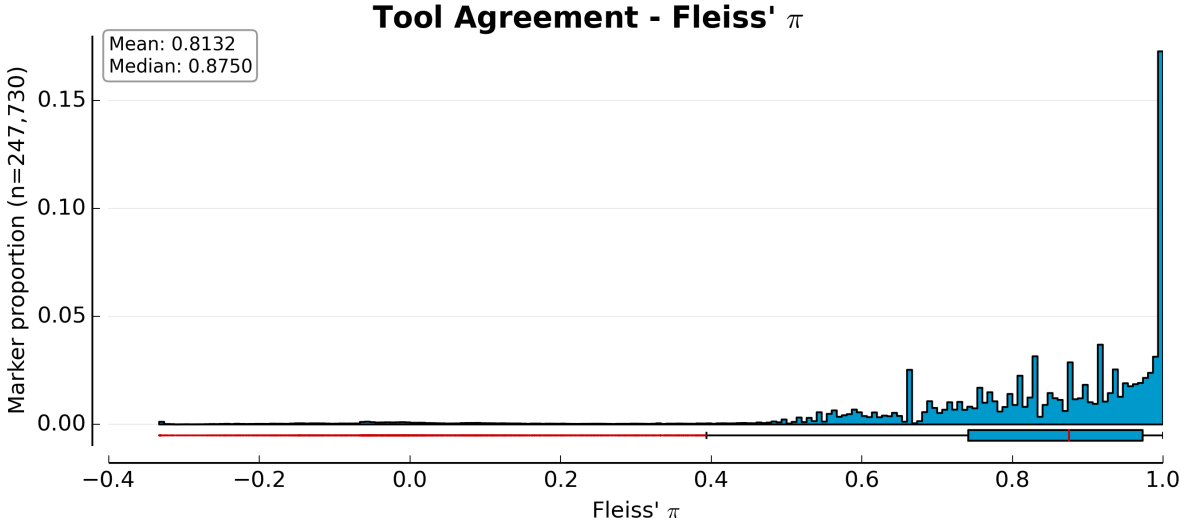


Figure 5.6.: Tool agreement. Distribution of Fleiss' π coefficient for all 247,730 markers when the following four tools are compared: *GenCall* (optimized cluster file), *GenoSNP* (optimized), *optiCall* (without excluding markers failing Hardy-Weinberg) and *zCall*. Values closer to 1 indicate good agreement. Using the interquartile range, there were a total of 12,122 outliers (represented as red dots in the boxplot).

Both metrics showed a high number of outliers according to the interquartile range dispersion [2]. To assess the overall agreement between the four tools (excluding *GenCall* using the original cluster file and the original version of *GenoSNP*), Fleiss' κ was computed (Figure 5.6). The agreement was fairly good between the tools when comparing all markers (common and rare). There was a total of 12,122 outliers according to the interquartile range dispersion, a majority of which (91.3%) were rare variants (MAF $< 1\%$, according to frequencies computed using *GenCall* results) and 5% were markers that were zeroed out while optimizing *GenCall*'s cluster file. The proportion of rare variants (when compared to common or zeroed out ones) was significantly higher in the outlier group (Wald statistic, $p < 0.001$; logistic regression). Out of the approximately three thousands outlier markers that were comparable with the *1000 Genomes Project* data, *GenCall* had the highest concordance for the three replicates of *NA12763* (average of 98.925%), followed by *zCall* (average of 98.448%) (Additional file 1: Table S3).

5.3.6. Error rate estimates

Error rates were estimated by using the genotypic model (Liu *et al.* [12]), where the six tools were evaluated (Figure 5.7). Except for the original version of *GenoSNP*, all tools showed a comparable estimated error rate mean (approximately 0.002). The *zCall* tool provided the lowest estimated error rate with a mean of 0.164%, followed by *GenCall* (optimized cluster file) with a mean of 0.166%. A non-parametric *Friedman Rank Sum* test comparing tools showed a significant difference in the error rate distributions (p-value $< 2.2 \times 10^{-16}$ in both cases). Pairwise dependent-samples non-parametric sign tests comparing pairwise tools were also significant (p-value $< 2.2 \times 10^{-16}$ in all cases).

5.4. Conclusions

This study compares the performance of widely used clustering tools when applied to genotyping data from Illumina's HumanExome BeadChip. This genotyping array includes a large proportion of

2. Outliers are observations that fall below $Q_1 - 1.5(IQR)$ or above $Q_3 + 1.5(IQR)$, where Q_1 and Q_3 are respectively the first and third quartiles and $IQR = Q_3 - Q_1$ is the interquartile range.

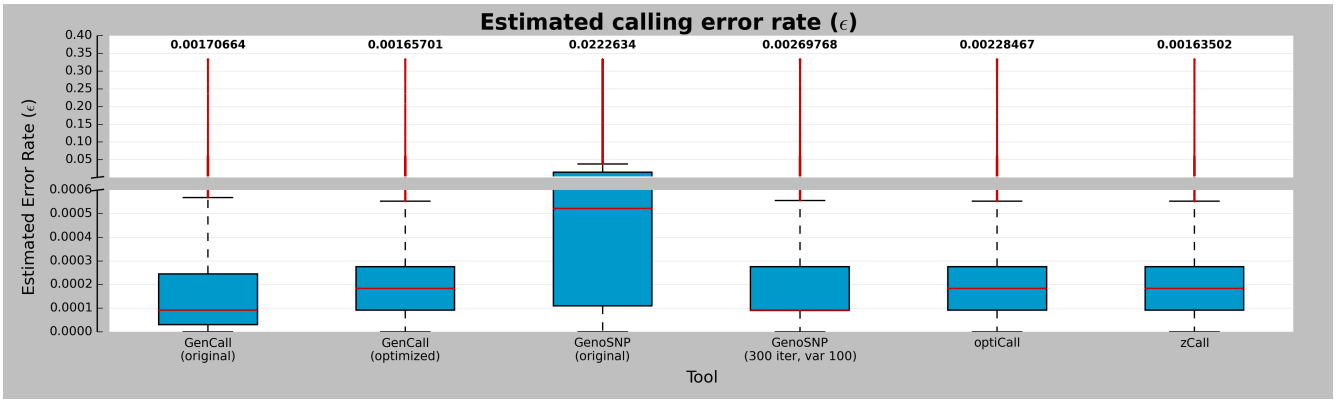


Figure 5.7.: Marker error rate distribution. Boxplots showing the marker error rate estimation for each the six tools: *GenCall* (original and optimized cluster files), *GenoSNP* (original and optimized), *optiCall* (without excluding markers failing Hardy-Weinberg) and *zCall*. The numbers at the top of the plot represent the mean of the error rate distribution and the red lines are the median of each distribution. Red dots represent the outliers according to the interquartile range. Values above 1/3 were discarded since it's not plausible to have such a large error rate in practice [12].

rare variants that were previously identified by sequencing technologies. The dataset used here included 10,520 unique samples along with a high number of technical replicates for quality assessment.

Contrary to our original expectations, *GenoSNP*, which relies on a *within-sample* model, did not perform well when used straight "out of the box". This might be explained by the high density of markers in each BeadPool, which is higher than for Illumina's Human 1M BeadChip (comprising an overall higher number of markers). The latter was successfully tested by *GenoSNP*'s authors. This problem was mostly obvious with cluster plots (Figure 5.1) and with graphs showing the summarized quality of the calls per sample (Figure 5.2). The concordance of results from the original *GenoSNP* tool with the *1000 Genomes Project*, however, remained high (mean of 87.87%). The optimized tool (modified initial variance and number of iterations) increased the concordance to a mean of 99.61%. The quality threshold of 0.8 provided a better separation of the three clusters, but increased the missing rate of both sample and marker (mean of 0.87%).

GenCall relies on a *between-sample* model that requires reference parameters to perform its clustering. As such, it is common practice to manually modify the reference cluster parameters to ensure the best quality of results when the population analyzed is different from the one used to generate

the reference parameters. This task requires a significant amount of manual labor which increases with the number of samples and markers. Loading the raw data, normalizing and modifying the original cluster file took one person 5 work days, compared to only a few days to generate the intensity files for the other tools. When we compared the genotype results from *GenCall* using the original cluster file with the ones generated with the optimized cluster file, we saw only limited improvement in the overall concordance of genotype calls with those of the gold standard. When partitioning calls according to their allele content, we saw a limited decrease in the concordance of homozygous calls involving the common allele when compared with the gold standard, while the concordance of calls involving the rare allele improved by approximately 0.5%. A limited improvement in the concordance between technical replicates was observed. However, the optimization of the cluster file had a greater impact on the missing rate per sample and per loci (0.19% improvement in both cases). According to normal quality control procedures, a filter to remove samples and markers with a missing rate greater than 2% is typically imposed prior to genetic analysis [144]. By optimizing the cluster file, an average of 2,963 markers (1.2%) per sample could be rescued by lowering the marker missing rate below the 2% quality threshold.

It should be noted that this study was limited by the lack of an independent gold standard for concordance analysis. *GenCall*'s (and *zCall*'s) concordance with the *1000 Genomes Project* might be overestimated due to the nature of its reference parameters. Indeed, the reference cluster file was created by using estimated cluster position for each of the markers in the HapMap population. Hence, the sample *NA12763* is expected to have a higher concordance value than samples from the Montreal Heart Institute Cohort owing to the CEU HapMap population. One possible way to overcome this limitation would be to sequence a few of our own cohort samples, however, it could be argued that the sequencing technology itself would not provide an adequate gold standard in this situation.

Many study designs will plan to include experimental replicates chosen from the genotyped cohort to assess the reproducibility of the genotyping pipeline and the precision of the results. Including

a reference sample to the study design offers the additional possibility of assessing the accuracy of the results. High precision is particularly important as it provides optimal power for statistical analysis [145] and can prevent type I errors due to plate biases or subgroup effects [146]. *GenCall* (using the original or the optimized cluster file) provided the highest concordance rate between experimental replicates (99.997%) when compared to other tools. The other tools (except for the original *GenoSNP*) performed relatively well, all providing a mean concordance greater than 99%.

All clustering tools had a high accuracy ($> 99.8\%$) when calling common markers (except for *GenoSNP* at only one of the three replicates of *NA12763*). Other metrics have shown that all the tools (once optimized) performed almost at the same efficiency level on the HumanExome BeadChip. The major difference arose when in the presence of rare markers, where the accuracy of all the tools decreased below 99.5% (as low as 96% for some) for genotypes involving the rare allele. *Within-sample* tools like *GenoSNP* can process samples in an efficient manner by running single samples and smaller batches of samples in parallel instead of having to wait for a large amount of samples to be genotyped and normalized. Unfortunately, even with the proper optimization of initial parameters, *GenoSNP* could not outperform the other tools. However, *GenCall*, Illumina's proprietary tool, performed better than the other tested tools with respect to concordance with the gold standard for genotypes involving the rare allele (accuracy) and slightly better for the concordance in between technical replicates (precision). The fact that *zCall* has been run as a post-process of *GenCall* without the use of the reference cluster file optimization (since it is not mandatory) might explain why it's accuracy was not as high as the optimized *GenCall* for calls involving the rare allele. Since the third version of *zCall* derives its thresholds from *GenomeStudio*'s report, the call rate will increase and better accuracy might be possible if the original cluster file is optimized beforehand.

Recommending a single clustering tool according to the metrics shown in this report is not straightforward. In general, *GenCall* (optimized cluster file) outperformed the other tools in terms of precision and accuracy (overall, and for the calls involving the rare allele). Its accuracy was also higher for the markers with low inter-tool agreement. However, when using the optimized cluster

file, *GenCall*'s accuracy for the homozygous calls (common allele) was lower than when using the default cluster file. When considering missing and estimated error rates, *zCall* outperformed the other tools, closely followed by *GenCall* (optimized cluster file). It is important to mention that the task of optimizing the cluster file is time consuming. Furthermore, all the other tools presented here require intensity data provided by the *GenomeStudio* software and possible file conversion, which increase the total execution time.

The parallel use of multiple clustering tools offers the possibility of identifying discordant markers which can be further investigated. But notably, the manual optimization of *GenCall*'s cluster file at those loci and the visual inspection of the cluster plots should provide high quality datasets for downstream analysis.

5.5. Competing interests

The authors declare that they have no competing interests.

5.6. Authors contributions

LPLP worked on the experimental design, optimized *GenoSNP*'s initial parameters, performed analyses, compared the tools, implemented the statistical tests and drafted the manuscript. MAL contributed to the analysis and to the manuscript. AB contributed to the statistical analyses and the manuscript. SP contributed to the experimental design, analysis and the manuscript. VN contributed to the manual optimization of *GenCall* and the manuscript. JCT leads the Montreal Heart Institute Cohort and directs the genomic project with the HumanExome BeadChip. MPD supervised the project, participated in the design and coordination of the study, supervised the laboratory and revised the manuscript. All authors read and approved the final manuscript.

5.7. Acknowledgements

This project was funded by the Montreal Heart Institute Foundation and by the Centre of Excellence in Personalized Medicine. We would like to acknowledge the team at the Beaulieu-Saucier Université de Montréal Pharmacogenomics Center who performed the genotyping of more than 10,000 samples of the Montreal Heart Institute Cohort on the HumanExome BeadChip together with all the participants of the cohort and the ones responsible for their recruitment.

5.8. Additional files

Additional file 1 - Additional Materials

Supplemental Equation S1: The genotypic model for error rate estimation was tested by Liu *et al.* for common variants only. However, we found that the possible values of ϵ were out of bound (*i.e.* negative or above one) for a majority of rare markers. For those cases, ϵ was approximated using $\epsilon \simeq (C_1 - C_3 + 1)/3$.

Supplemental Table S1: Distribution of n samples by calling tool in q categories. The set of possible categories are all possible genotypes (*i.e.* $q \in \{\mathrm{AA}, \mathrm{AB}, \mathrm{BB}, 00\}$, where 00 represents the *no call* category). This table is computed for each marker and for each pair of calling tools. The overall agreement probability and Cohen's κ are shown in Equation 5.1 and 5.2 of the main text, respectively.

Supplemental Table S2: Distribution of r calling tools by n samples and q response categories. The set of possible categories are all possible genotypes (*i.e.* $q \in \{\mathrm{AA}, \mathrm{AB}, \mathrm{BB}, 00\}$, where 00 represents the *no call* category). This table is computed for each marker and for each calling tool. Fleiss' π is explained in Equation 5.3 of the main text.

Supplemental Table S3: Call concordance and number of compared markers for the three control replicates when compared to the *1000 Genomes Project* for the markers that were outliers for their Fleiss' π values. The following four tools were compared: *GenCall* (optimized cluster file), *GenoSNP* (optimized), *optiCall* (without excluding markers failing Hardy-Weinberg) and *zCall*.

6. Partitioning of copy-number genotypes in pedigrees

Louis-Philippe Lemieux Perreault a travaillé sur la méthodologie de *CNGen*, a implémenté *CNGen* ainsi que l'outil d'accompagnement, a effectué la validation de l'algorithme en utilisant des simulations et a écrit le manuscrit. Géraldine Asselin a participé à la validation de l'algorithme et a aidé à la rédaction du manuscrit. Gregor U Andelfinger a participé à la conception de l'étude et a aidé à la rédaction du manuscrit. Marie-Pierre Dubé a participé au design et à la coordination de l'étude, a produit la méthodologie de *CNGen* et a aidé à la rédaction du manuscrit. Tous les auteurs ont lu et ont approuvé le manuscrit final.

BMC Bioinformatics 2010, **11**:226; doi:10.1186/1471-2105-11-226

Partitioning of copy-number genotypes in pedigrees

Louis-Philippe Lemieux Perreault[*,1,2], Gregor U Andelfinger[2,3], Géraldine Asselin[1] and Marie-Pierre Dubé[*,1,2].

[1]Montreal Heart Institute Research Center, 5000, Bélanger Street, Montréal, Canada

[2]Université de Montréal, 2900, chemin de la tour, Montréal, Canada

[3]CHU Sainte-Justine, 3175, Chemin de la Côte-Sainte-Catherine, Montréal, Canada

Abstract

Background: Copy number variations (CNVs) and polymorphisms (CNPs) have only recently gained the genetic community's attention. Conservative estimates have shown that CNVs and CNPs might affect more than 10% of the genome and that they may be at least as important as single nucleotide polymorphisms in assessing human variability. Widely used tools for CNP analysis have been implemented in *Birdsuite* and *PLINK* for the purpose of conducting genetic association studies based on the unpartitioned total number of CNP copies provided by the intensities from Affymetrix's Genome-Wide Human SNP Array. Here, we are interested in partitioning copy number variations and polymorphisms in extended pedigrees for the purpose of linkage analysis on familial data.

Results: We have developed *CNGen*, a new software for the partitioning of copy number polymorphism using the integrated genotypes from *Birdsuite* with the Affymetrix platform. The algorithm applied to familial trios or extended pedigrees can produce partitioned copy number genotypes with distinct parental alleles. We have validated the algorithm using simulations on a complex

pedigree structure using frequencies calculated from a real dataset of 300 genotyped samples from 42 pedigrees segregating a congenital heart defect phenotype.

Conclusions: *CNGen* is the first published software for the partitioning of copy number genotypes in pedigrees, making possible the use CNPs and CNVs for linkage analysis. It was implemented with the *Python* interpreter version 2.5.2. It is compatible with current Linux, Windows and Mac OS workstations.

6.1. Background

Copy number variations (CNVs) and polymorphisms (CNPs) have recently gained interest as a novel tool to study the relationship between genomic variation and disease. CNVs and CNPs are widespread throughout the genome and were shown to be largely heritable while being responsible for a significant amount of inter-variability in human [1]. They can also appear *de novo* at a significant rate, both in germline and somatic cells [49]. Any variation in copy number has the possibility of affecting a wide spectrum of genes, which might lead to genomic disorders [4]. Variation in gene-expression levels can occur for genes located within a region of copy number variation [49], and negative correlations between CNV and gene expression were reported in approximately 10% of cases [147]. It is currently estimated that up to 12% of the genome is subject to copy number variations [46, 47]. Those genetic variations are likely to play an important role in the etiology of common disease and sporadic birth defects [1], partly attributable to their higher mutation rate as compared to point mutations [52] and due to their considerable size.

High-density SNP genotyping arrays are commonly used for CNV/CNP analysis. Those arrays provide signal intensities of alleles across all SNPs which can be used to infer copy numbers along with a selection of CNV-specific probes. The presence of a CNV/CNP region has the potential to confuse SNP calling algorithms if unaccounted for, as SNPs can be represent with multiple or single alleles. It is then crucial to gain knowledge of CNV and CNP in genetic analysis, even when using SNPs as a marker.

While amenable to genetic association studies, the use of CNVs and CNPs in linkage analysis with multi-generational family data has up to now been greatly limited by the requirement of chromosome-specific copy number assignments, which, to our knowledge, none of the current software indexed in the literature is able to provide. Multi-allelic partitioned copy number polymorphisms have the potential to offer a new and powerful tool for linkage analysis. Today's high density SNP panels offer near-optimal coverage for linkage analysis. However, some regions, especially those with copy number polymorphisms, may have been less well covered due to the requirements of Mendelian consistency prior to linkage analysis. Although representing only a minute fraction of the genome, the partitioning of copy number genotypes has the potential to help fill-in the remaining linkage coverage gaps.

The use of genome-wide association studies (GWAS) with unrelated cases and controls is a popular approach for the discovery of genetic variants responsible for common genetic diseases [148]. Linkage analysis with extended pedigrees is of limited use for the identification of common polymorphisms of low effect, but it does offer high detection power with more penetrant variants even in the presence of multiple rare causal variants at a single locus [149] or highly penetrant rare variants throughout the genome. Furthermore, the combined use of pedigree-based linkage analysis and association studies in a multistage approach was argued by Elston *et al.* to be both powerful and advantageous [149]. Significantly linked markers can emphasize candidate genes for subsequent association study and information on candidate loci can be incorporated into association tests using either a generalized logistic regression [150] or a quantitative linkage score [151].

Here, we are interested in using CNV and CNP data from the Affymetrix 6.0 chip analyzed with the *Fawkes* program of the *Birdsuite* software [65, 97]. *Fawkes* creates an integrated genotype from SNPs, rare copy number variations and common copy number polymorphisms genotypes information, providing the number and type (A or B) of each allele for each SNP on the Affymetrix Genome-Wide Human SNP Array 5.0 and 6.0 chips. While the suite comes with *Python* scripts for file compatibility with the whole-genome association toolset PLINK [128], no software is available to

conduct chromosome assignment of the copy number genotypes based on pedigree information. We propose a new algorithm called *CNGen* that uses SNP genotypes in multi-generational pedigrees to convert *Fawkes*' genotypes into partitioned copy number genotypes (CN genotypes) which can then be treated as multi-allelic markers by common linkage software such as *MERLIN* [152]. We have developed *Python* scripts to encode CN genotypes into multi-allelic genotypes. We have validated and successfully applied the algorithm in the analysis of multi-generational pedigrees through simulation procedures.

6.2. Implementation

The standard *Fawkes* output file is tabulated with samples in columns and probe sets (SNPs) in rows. Each cell contains a *Fawkes* call that is a comma-separated value of the form $[a, b]$ where a is the number of copies of allele A and b, the number of copies of allele B. Five different *Fawkes* calls are possible:

1. undefined calls, from a probe set of the form $[-1, -1]$ that was unresolved by *Fawkes*;
2. heterozygous calls of the form $[a, b]$ where a and $b \neq 0$;
3. null calls of the form $[0, 0]$, representing a null genotype;
4. hemizygous calls of the form $[0, b]$ or $[a, 0]$, where a and $b = 1$;
5. homozygous calls, of the form $[0, b]$ or $[a, 0]$, where a and $b > 1$.

CNGen converts *Fawkes* calls into partitioned CN genotypes as comma-separated values of the form $[T_1m, T_2n]$ where T_i is the allele type (one of A, B or N for null) on one of the parental chromosome, and m and n represent the number of copies of the named allele type on the specified chromosome. The N allele type represents an absence of either an A or B allele on a given parental chromosome. The partitioning of copy numbers is accomplished according to the rules of Mendelian transmission and under the general assumption that ancestral copy number expansions were of the same allele type, *i.e.* a copy number expansion from 1 to 2 copies is not allowed to bear both A and B alleles on the same chromosome strand.

Table VI.I.: Direct Fawkes conversion. Example of direct conversion from integrated genotypes (*Fawkes* to CN genotypes). The type-1 homozygous genotypes are converted using information from first-degree relatives with one of those *Fawkes* calls: heterozygous, null or hemizygous.

***Fawkes* genotypes**		**CN genotypes**
	Undefined	
$-1,-1$	$\longrightarrow$	$-1,-1$
	Heterozygous	
a, b (a and $b \neq 0$)	$\longrightarrow$	Aa, Bb
	Null	
$0, 0$	$\longrightarrow$	N, N
	Single hemizygous	
$1, 0$	$\longrightarrow$	$A1, N$
$1, 0$	$\longrightarrow$	$A1, N$

This last assumption affects only copy numbers of two or more, since single-copy alleles will result in one copy which will by default be located on a single chromosome. Situations with two copies where the true CN genotype is $[A2, N]$, $[B2, N]$ and $[A1, B1]$ will be appropriately called. However true $[A1B1, N]$ will not and will likely give rise to Mendelian inconsistencies which will be coded as undefined by the *CNGen* algorithm. Expansions beyond 2 copies were found less frequently than 0, 1, and 2 copies by a survey of 300 genotyped individuals in 42 pedigrees presenting a congenital heart defect phenotype. Overall, only 0.07% of *Fawkes* calls had three or more copies (expansions), compared to 2.2% with 0 or 1 copy (deletions) and 97.6% with 2 copies, with the rest being undefined *Fawkes* calls.

6.2.1. Step 1 - Partitioning of non-homozygous calls

The algorithm begins by parsing the *Fawkes* calls to generate in this first pass the CN genotypes for the first four of the five possible *Fawkes* calls. Undefined and null *Fawkes* genotypes are set to undefined or null CN genotypes, respectively. For single hemizygous *Fawkes* genotype, the first chromosome is set to hold the deletion (N) and the other, the given allele (A or B). Finally, heterozygous *Fawkes* calls are partitioned such that each chromosome receives the copies of only one allele type. Those conversions from *Fawkes* genotypes to partitioned CN genotypes are summarized in Table VI.I.

6.2.2. Step 2 - Partitioning of type-I homozygous calls

We distinguish two types of homozygous *Fawkes* calls based on the genotype conversion method used: type-I and type-II. CN genotype partitioning for type-I homozygous *Fawkes* calls is solved by relying on information from a heterozygous first-degree relative and assuming Mendelian transmission. The algorithm searches for heterozygous first-degree relatives (parents, children and siblings) of the index individual to be converted (I), as those will have partitioned CN genotypes that can be used as reference. Figure 6.1 presents the different scenarios for type-I homozygote partitioning.

Step 2a

When a parent of I has a heterozygous CN genotype of the form $[Am_p, Bn_p]$, then I is assigned the following CN genotype:

$$I_{CN,L} = \begin{cases} [Am_p, Aa - m_p] & \text{if } I_{F,L} = [a, 0] \\ [Bn_p, Bb - n_p] & \text{if } I_{F,L} = [0, b] \end{cases} \tag{6.1}$$

where $I_{F,L}$ and $I_{CN,L}$ are the *Fawkes* genotype and the new CN genotype of the index individual at locus L, respectively (Figure 6.1A). If $a - m_p$ or $b - n_p$ equals 0, the second partitioned CN allele becomes N.

Step 2b

If I does not have a heterozygous parent, the algorithm searches for the presence of a child with a heterozygous CN genotype of the form $[Am_c, Bn_c]$. The partition of the CN genotype is solved as presented in Equation (6.1) by replacing m_p and n_p by m_c and n_c respectively (Figure 6.1B).

Step 2c

If I does not have a heterozygous child, then the algorithm searches for the presence of two siblings with distinct heterozygous CN genotypes $[Am_{s1}, Bn_{s1}]$ and $[Am_{s2}, Bn_{s2}]$ for which the cardinality of the pool of CN alleles of the same type as I is two, *i.e.* ($|\{Am_{s1}, Am_{s2}\}| = 2$ if $I_{F,L} = [a, 0]$,

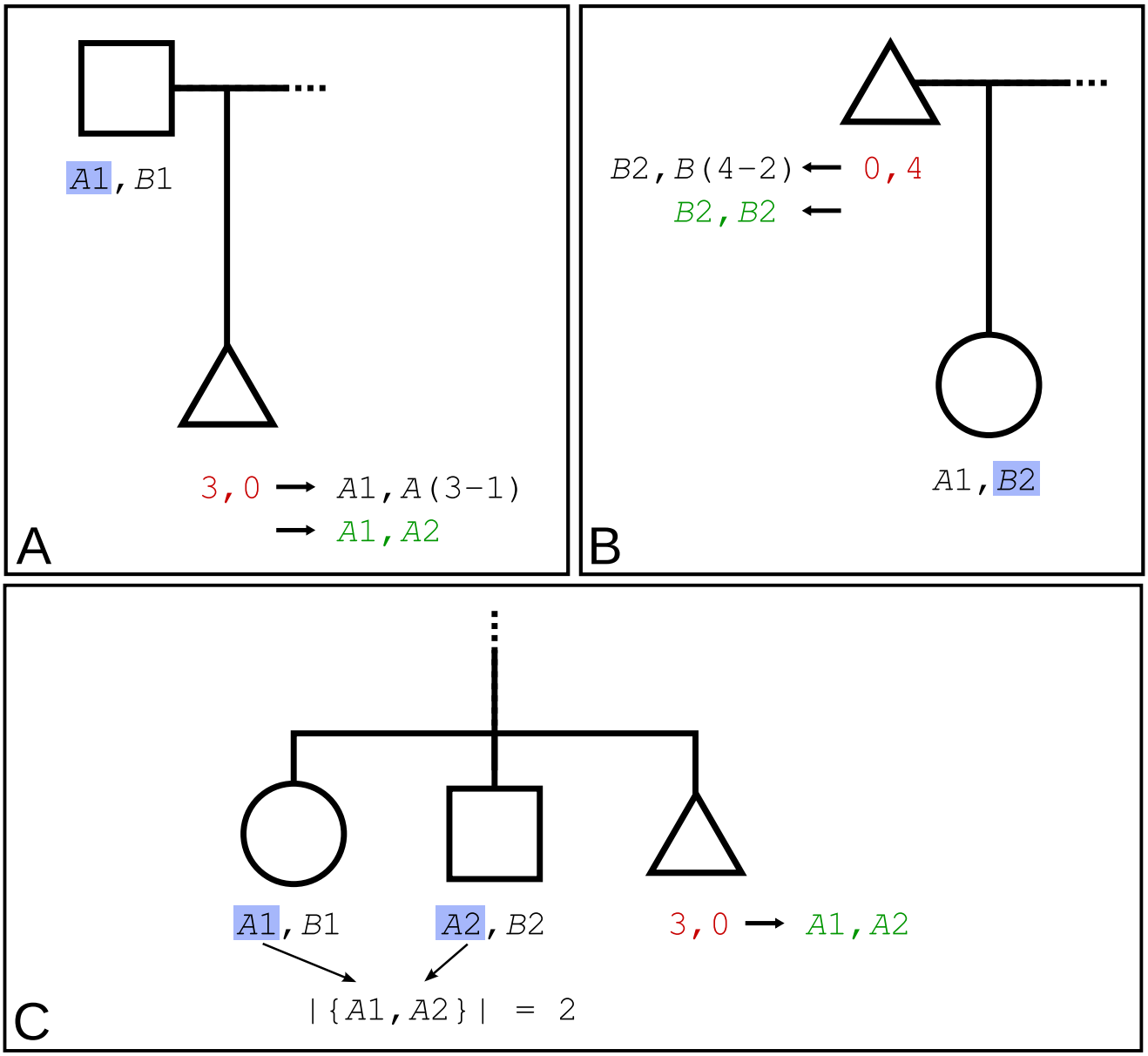


Figure 6.1.: Type-I homozygous conversion. Representation of the conversion of Type-I homozygous *Fawkes* genotypes (red) into partitioned CN genotypes (green). The triangle represents the index individual (I). (A) Conversion of I's *Fawkes* genotype using one heterozygous parent, (B) one heterozygous child or (C) two heterozygous siblings. In each case, the sex of the individual used for conversion is not important.

$m_{s1} \neq m_{s2}$) or ($|\{Bn_{s1}, Bn_{s2}\}| = 2$ if $I_{F,L} = [0, b]$, $n_{s1} \neq n_{s2}$). Then, I is assigned the following CN genotype:

$$I_{CN,L} = \begin{cases} [Am_{s1}, Am_{s2}] & \text{if } I_{F,L} = [m_{s1} + m_{s2}, 0] \\ [Bn_{s1}, Bn_{s2}] & \text{if } I_{F,L} = [0, n_{s1} + n_{s2}] \\ \text{do nothing} & \text{otherwise} \end{cases} \tag{6.2}$$

Restricting the conditions $m_{s1} \neq m_{s2}$ or $n_{s1} \neq n_{s2}$, ensures that both CN alleles originate from the two distinct parents (Figure 6.1C). Any *Fawkes* homozygous calls that remain un-converted are then flagged as type-2 *Fawkes* homozygous calls and the algorithm proceeds to step 3.

6.2.3. Step 3 - Partitioning of type-II homozygous calls

CN genotype partitioning of type-II homozygous *Fawkes* calls proceeds by assuming Mendelian transmission of CN alleles and by relying on information in the nuclear pedigree of I. The algorithm searches for a solution according to the following sequential attempts. Figure 6.2 presents the different scenarios for type-II homozygous partitioning.

Step 3a

First, the algorithm searches for the presence of one parent of I that is homozygous for a CN genotype of the same allele-type as I such as $[T_1 m_p, T_2 n_p]$ where $T_1 = T_2$, $T_{i=1,2} \in \{A, B, N\}$ and $m_p = n_p$; in which case I is assigned its CN genotype according to Equation (6.1) (Figure 6.2A).

Step 3b

Analogously to step 3a above, a child of I presenting a homozygous CN genotype $[T_1 m_c, T_2 n_c]$ can be used according to Equation (6.1) (m_c replacing m_p and n_c, n_p) (Figure 6.2B).

Step 3c

If no such parent or child exists, the algorithm then searches for the presence of two siblings, one with a heterozygous genotype of the form $[Am_{s1}, Bn_{s1}]$, and the other one with a homozygous CN

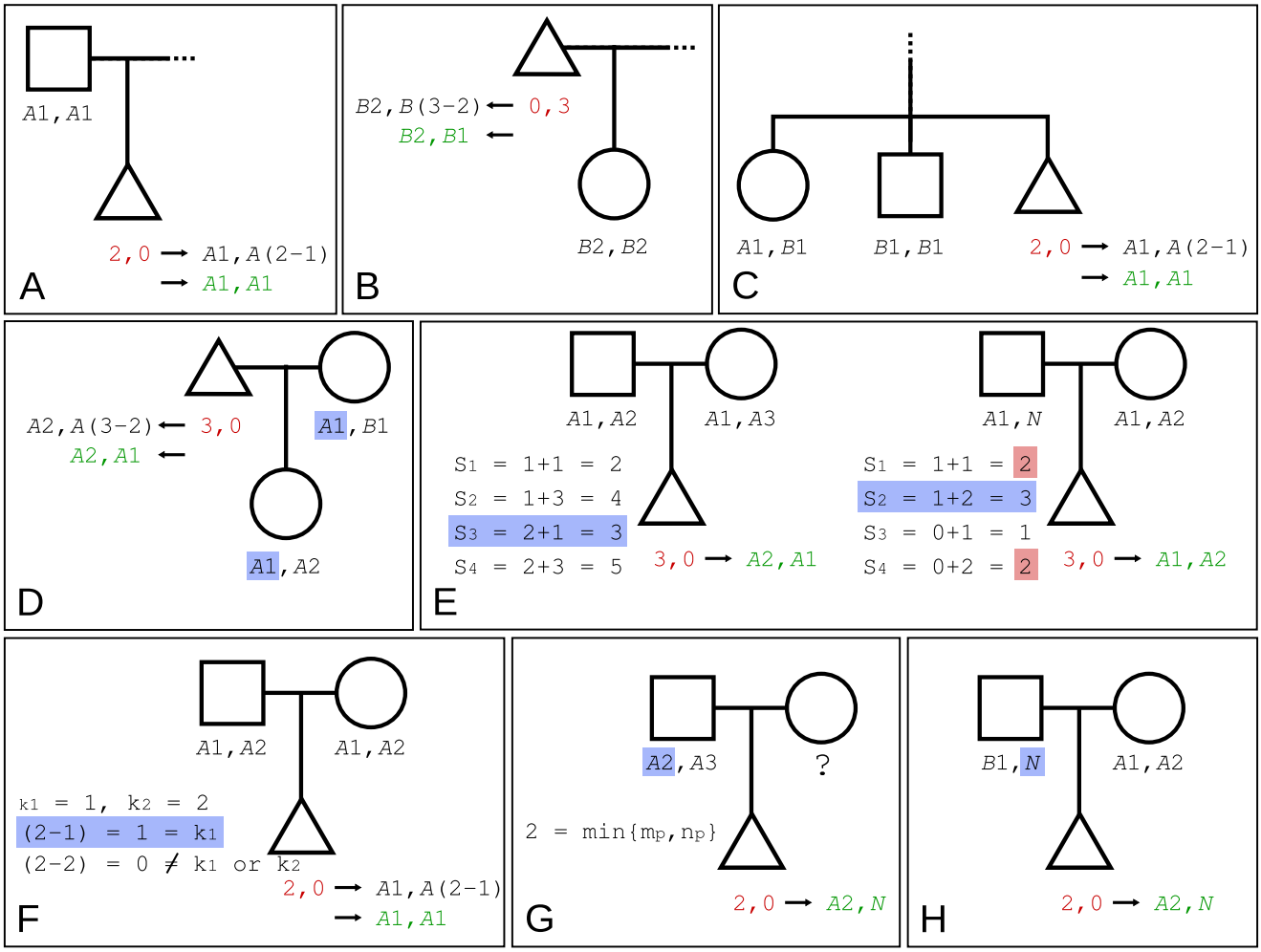


Figure 6.2.: Type-II homozygous conversion. Representation of the conversion of Type-II homozygous *Fawkes* genotypes (red) into partitioned CN genotypes (green). The triangle represents the index individual (I). (A) Conversion of I's *Fawkes* genotype using one homozygous parent or (B) using one homozygous child. (C) represents the conversion using two sibs: one heterozygote and one homozygote. (D) uses a child and the spouse of the index individual. (E) uses the two homozygous parents of I. The pedigree on the left shows the conversion when there are 4 different sums, and the pedigree on the right, when there are only 3 different sums (see step 3e above). Panel (F) shows the conversion when two homozygous parents with the same genotype is used. Finally, the conversion methods when one parent has a unknown genotype or a null allele are shown in panel (G) and (H) respectively.

genotype with identical CN alleles excluding null alleles and of a different allele-type than that of I (*i.e.* $[Am_{s2}, An_{s2}]$ if I's *Fawkes* genotypes is $[0, b]$ or $[Bm_{s2}, Bn_{s2}]$ if I's *Fawkes* genotypes is $[a, 0]$). In this case, $I_{CN,L}$ is defined by Equation (6.1) with replacement of m_p and n_p by m_{s1} and n_{s1}, respectively (Figure 6.2C).

Step 3d

If I's genotype at the given locus remains unconverted, the algorithm searches for the presence of one child and the spouse of I where both have distinct CN genotypes. In this case, one CN allele is obligatorily shared between the child and the spouse and the remaining CN allele of the child can be assigned to I. The spouse's CN genotype has the form $[T_1 m_s, T_2 n_s]$ and the child's CN genotype $[T_1 m_c, T_2 n_c]$ where $|\{T_1 m_s, T_2 n_s\} \cap \{T_1 m_c, T_2 n_c\}| = 1$. I is assigned the remaining child's CN allele according to $\{T_1 m_c, T_2 n_c\} - \{T_1 m_s, T_2 n_s\}$ and the algorithm infers the other CN allele of I (Figure 6.2D).

Step 3e

If the algorithm has not yet converted the *Fawkes* genotype according to the above steps, it then searches for cases where the two parents of I are both homozygotes of the same allele type as I but with distinct CN genotypes. Here, one parent's genotype can be inferred if its CN genotype is undefined. A solution exists if the first parent has a CN genotype of the form $[T_1 m_{p1}, T_2 n_{p1}]$ and the second parent, $[T_1 m_{p2}, T_2 n_{p2}]$, where $m_{p1} \neq m_{p2}$ or $n_{p1} \neq n_{p2}$. In both cases, $T_1 = T_2 = A$ if $I_{F,L} = [a, 0]$ and $T_1 = T_2 = B$ if $I_{F,L} = [0, b]$. Parents may have a null allele on one chromosome. The following sums are then calculated:

$$
\begin{aligned}
s_1 &= m_{p1} + m_{p2} \\
s_2 &= m_{p1} + n_{p2} \\
s_3 &= n_{p1} + m_{p2} \\
s_4 &= n_{p1} + n_{p2}
\end{aligned}
$$

If the number of unique sums is 4 (*i.e.* $|\{s_1, s_2, s_3, s_4\}| = 4$), the sum s_i that corresponds to the *Fawkes* genotype of I is used to assign the corresponding parental CN alleles to I. If the number of unique sums is 3 (*i.e.* $|\{s_1, s_2, s_3, s_4\}| = 3$), then the algorithm checks whether I's *Fawkes* genotype matches the min or $\max\{s_1, s_2, s_3, s_4\}$, in which case the corresponding parental CN alleles can be assigned to I (Figure 6.2E).

Step 3f

If the two homozygous parents have identical CN genotypes of the same allele-type as that of the index individual (first parent having a CN genotype $[T_1 m_{p1}, T_2 n_{p1}]$ and second parent, $[T_1 m_{p2}, T_2 n_{p2}]$ where $m_{p1} = m_{p2} = k_1$ and $n_{p1} = n_{p2} = k_2, T_1 = T_2 = A$ if $I_{F,L} = [a, 0]$ or $T_1 = T_2 = B$ if $I_{F,L} = [0, b]$ for both parent), then I is assigned the CN genotype described in Equation (6.3) (Figure 6.2F).

$$I_{CN,L} = \begin{cases} [T_1 k_1, T_2(a+b) - k_1] & \text{if } (a+b-k_1) \in \{k_1, k_2\} \\ [T_1 k_2, T_2(a+b) - k_2] & \text{if } (a+b-k_2) \in \{k_1, k_2\} \end{cases} \tag{6.3}$$

Step 3g

If only one parent of I has a CN genotype with the same CN allele type as I ($[T_1 m_p, T_2 n_p]$ where $T_1 = T_2 = A$ if $I_{F,L} = [a, 0]$ or $T_1 = T_2 = B$ if $I_{F,L} = [0, b]$) with the possibility of one null allele, if $(a + b) = \min\{m_p, n_p\} = z$, then $I_{CN,L} = [T_1 z, N]$ (Figure 6.2G).

Step 3h

Finally, if one parent of I has a heterozygous CN genotype containing a N CN allele and the remaining allele is of a different allele-type as that of I (*i.e.* $[Bn_p, N]$ if $I_{F,L} = [a, 0]$ or $[Am_p, N]$ if $I_{F,L} = [0, b]$) and if both parents of I have a CN genotype for this loci and the trio respects Mendelian transmission (or only one parent is converted or genotyped), the N allele is assigned to I and the second allele is inferred (Figure 6.2H).

6.2.4. General procedure of the algorithm

The algorithm developed reads a *pedfile* in linkage format containing the pedigree structures, and then opens the *Fawkes* output file generated by *Birdsuite*. Reading-in one marker of the *Fawkes* file at a time, *CNGen* begins by converting *Fawkes* genotypes of type 1 to 4 into CN genotypes as described in step 1 (see Table VI.I). Next, homozygous type-I calls are converted based on heterozygous first-degree relatives (step 2 and Figure 6.1). Any encountered Mendelian inconsistencies are reported. Unconverted type-I homozygous calls are flagged as type-II. Then, the algorithm attempts to partition the remaining type-II homozygous calls by inspection of the converted first-degree relatives of the index individual according to procedures described in step 3 (Figure 6.2). Following Mendelian laws, and based on first degree relatives' CN genotypes, obligate genotype assignments are resolved. The algorithm cycles to resolve all unconverted type-II homozygote each time it has successfully partitioned at least one call. When no more calls can be partitioned, remaining *Fawkes* calls and obligate Mendelian inconsistencies are set to a CN genotype of $[-2, -2]$ and the algorithm proceeds to the following marker.

The algorithm outputs a tabulated file containing partitioned CN genotypes following the *Fawkes*' format. A log file is created and summary statistics of the partitioning procedures are sent to the standard output, including the percentage of each type of calls, the percentage of successful conversions and the number of Mendelian inconsistencies found during the process. *CNGen* does not specifically search for all Mendelian errors in the pedigrees but it reports those found during type-I and -II homozygous call conversions (step 2 and 3, respectively). The popular program *PedCheck* [153] can be used to systematically search for Mendelian errors, as per common linkage practice. A companion tool to interface with *PedCheck* was developed.

6.3. Results and Discussion

6.3.1. Implementation

The *CNGen* algorithm was implemented with the *Python* interpreter version 2.5.2. It was tested successfully on current Linux, Windows and Mac OS workstations. System resource requirements are dependent on the size of the input datasets, proportionally with the number of samples in the analysis. On a modern Linux workstation (Intel® Xeon® CPUs @ 2.40GHz), the conversion of approximately 273 million calls (909,622 markers from the Affymetrix 6.0 chip for 42 pedigrees [300 individuals]) required less than 10 Mb of RAM and a little more than one hour of computation time. *CNGen* is the first software to produce partitioned copy number genotype from *Birdsuite*'s integrated SNP genotypes. Partitioned CN genotypes offer the valuable possibility of using copy number variation in the context of linkage studies.

6.3.2. Validation

We have validated the algorithm using simulations on a multi-generational pedigree consisting of 47 individuals including 14 founders (Figure 6.3). Gene-dropping simulations were generated. First, founders were assigned a null, hemizygote, heterozygote or homozygote CN genotype state following proportions given by real data (∼0.0337%, ∼2.14%, ∼26.9% and ∼70.9%, respectively). An allele is then randomly chosen from a set of all possible CN genotypes depending of the given state. Mendelian segregation laws were used to assign CN genotypes to non-founding pedigree members, receiving one random allele from each parent. 1% of all CN genotypes were randomly selected and recoded as undefined CN genotypes ($[-1,-1]$). CN allele frequencies are presented in Table VI.II. CN genotypes were then converted into *Fawkes* genotypes based on the number of A and B alleles (*i.e.* $[Am, Bn] \longrightarrow [m, n]$, $[Am, An] \longrightarrow [(m+n), 0]$, $[Bm, N] \longrightarrow [0, m]$, etc.). Finally, *CNGen* was used to partition the *Fawkes* genotype back into CN genotypes and comparison between the true CN genotypes and the ones inferred by *CNGen* were compared. Three million validation runs were thus completed, for which more than 140 million genotype conversions were made, and which covered every possible conversion step from *Fawkes* to CN genotypes (additional

Table VI.II.: Allele frequencies after simulation. CN allele frequencies after three million simulations, frequencies for 14 founders and all 47 pedigree members are presented.

Alleles	Only Founders		All individuals	
	n	%	n	%
$A1$	12294813	0.146	41275221	0.146
$A2$	10728256	0.128	35999291	0.128
$A3$	6661654	0.0793	22367164	0.0793
$A4$	4068729	0.0484	13667675	0.0485
$A5$	1472417	0.0175	4938921	0.0175
$B1$	12291382	0.146	41262843	0.146
$B2$	10732261	0.128	36029869	0.128
$B3$	6662835	0.0793	22364446	0.0793
$B4$	4068582	0.0484	13652583	0.0484
$B5$	1473733	0.0175	4949877	0.0176
N	12706756	0.151	42672446	0.151
-1	838582	0.00998	2819664	0.01
Total	**84000000**		**282000000**	

file 1). The validation procedure confirmed that all converted genotypes by *CNGen* were accurate. Irresolvable homozygous type-II calls due to lack of information from first-degree relatives were checked and validated.

For an additional 30,000 validation runs (additional file 2), we substituted the CN genotype of a random pedigree member with a different randomly selected CN genotype and allowed 10% of CN genotypes to be recoded as undefined CN genotypes ($[-1, -1]$). Following *CNGen*, we ran *PedCheck* and nuclear families where inconsistent transmissions were found were set to missing. Overall, 81% of the inserted CN genotype errors were detected by the process. 59% of 30,000 simulations resulted in the concerned nuclear family being detected by *PedCheck*. In 22% of the 30,000 simulation runs, *CNGen* had assigned an undefined CN genotype at the modified individual. *CNGen* and *PedCheck* assigned an undefined value to respectively 14% and 7% of the 1,410,000 calls for a total of 21% of undefined calls. Overall, only 5,506 out of the 30,000 simulations (18.4%) resulted in a wrong CN genotype assignment to the substituted individual or to his first-degree relatives, representing an undetected genotype error rate of $\sim$0.5% for 1,410,000 calls (30,000 simulations $\times$ 47 individuals) with a simulated 2.13% genotyping error rate. In a typical study exposed to a 1% genotyping error

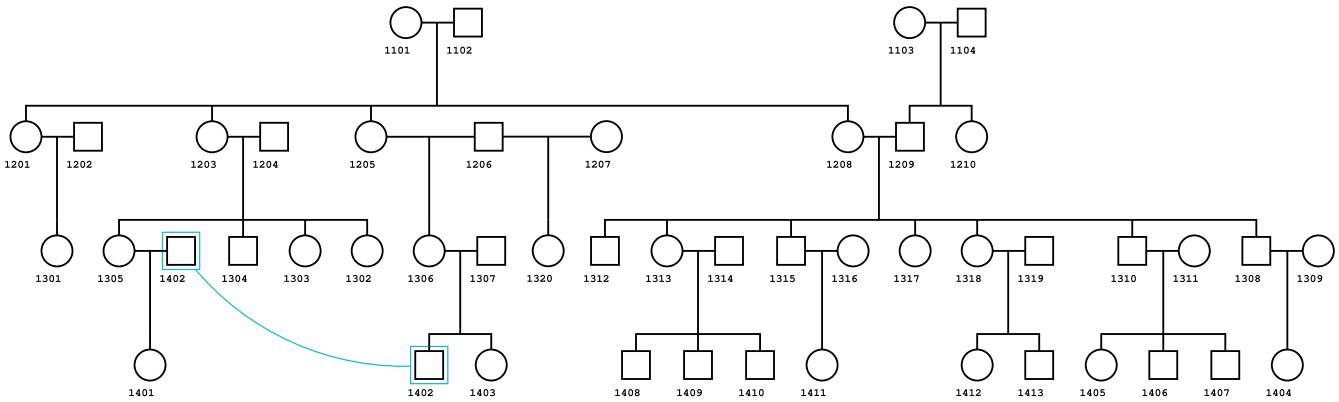


Figure 6.3.: Complex pedigree for the simulation. Representation of the complex pedigree used for validation simulation runs. The pedigree has 47 individuals including 14 founders. Individual *1402* creates a consanguinity loop in the pedigree. The diagram is modified from *Cranefoot*'s resulting pedigree [13].

rate, this would result in 0.2% of undetected genotype errors. These findings confirm that *CNGen* will not result in an excess of false calls in the presence of erroneous or *de novo* CNP.

6.4. Conclusions

CNGen is, to our knowledge, the first software that allows the partitioning of copy number genotypes in extended pedigrees for the purpose of linkage analysis with CNPs. *CNGen* is a flexible, open source *Python* program that can process integrated SNP genotypes from the *Fawkes* routine of the *Birdsuite* program for high-density SNP genotyping arrays. *Birdsuite* was developed for the Affymetrix's SNP array 5.0 and 6.0, but, as mentioned by the *Birdsuite* authors, the concepts and approach can be applied to any genotyping array [65] and they are planning on providing support for other high-throughput genotyping platforms, such as the Illumina 1M.

The *CNGen* algorithm is not limited to the *Fawkes* procedure. As long as the input file format is respected, *CNGen* will conduct the partitioning process. For instance, results from the *PennCNV* software [74] could be used.

The *CNGen* algorithm relies upon the assumption that ancestral copy number expansions are of the same allele type on a given chromosome. In a recent publication by Hastings *et al.* [49], a

general overview of the molecular mechanisms of change in gene copy number was presented, owing strong support for the involvement of DNA repair mechanisms which would, in great majority, be concordant with chromosome-specific expansions. There is a range of possibilities however, and copy number expansions occurring during recombination at meiosis, for example, could lead to different allele-type CN expansions. For regions where the assumption of identical allele-type in expansions doesn't hold, the majority will lead to Mendelian inconsistencies following the partitioning algorithm, and will be removed during data quality controls. This will result in a lower number of partitioned genotype for linkage analysis.

Our simulation experiments support the validity of the *CNGen* algorithm and its robustness to *Fawkes* genotype errors and *de novo* mutations.

6.5. Availability and requirements

Project name: CNGen

Project home page: http://www.statgen.org/ in the download section

Operating system(s): Platform independent

Programming language: Python™

Other requirements: Standard Python Software 2.5 or 2.6

License: none

Any restrictions to use by non-academics: none

6.6. Authors' contributions

LPLP worked on the methodology of *CNGen*, implemented *CNGen* and the companion software, performed the validation of the algorithm using simulations and drafted the manuscript. GA participated in the validation of the algorithm and helped to draft the manuscript. GUA participated in the conception of the study and helped to draft the manuscript. MPD conceived of the study,

participated in its design and coordination, produced the methodology behind *CNGen* and helped to draft the manuscript. All authors read and approved the final manuscript.

6.7. Acknowledgements

Funding was provided by the Canadian Institutes of Health Research (CIHR) and Heart and Stroke Foundation of Canada (GMHD 79045). MPD is supported by the FRSQ (Fonds de la recherche en santé du Québec).

6.8. Additional Files

Additional file 1 - 250_thousand_validations.tar.bz2

Archive containing the simulated data for 250 thousand runs (out of 3 million) on a pedigree containing 47 individuals (14 founders). The archive contains the simulated *Fawkes*' calls (file `validation.fawkes_calls`), the partitioned genotyped compute by *CNGen* and the corresponding log file (file `cn_genotype_calls_validation` and `CNGen.log`, respectively) and the pedfile corresponding to the complex pedigree used for simulation (file `pedfile.txt`). The file (18 Mb) has been uploaded with the present document, and is also available at http://statgen.org/downloads.

Additional file 2 - 3_thousand_validations_with_errors.tar.bz2

Archive containing the simulated data for 3 thousand runs with Mendelian errors. The archive contains the same file structure as the first additional file. The data has been splited into three files because of *PedCheck*'s limitations. The file (2.2 Mb) has been uploaded with the present document, and is also available at http://statgen.org/downloads.

7. Methodological Challenges in Genome-wide Association Studies of Rare Copy Number Variants

Eleni Giannoulatou et Louis-Philippe Lemieux Perreault ont contribué conjointement et de manière équivalente au projet décrit dans le manuscrit. Ils ont participé au design expérimental de l'étude, au nettoyage des données, aux analyses statistiques et ont écrit le manuscrit. Eleni Giannoulatou s'est occupé de l'analyse des données provenant des micropuces d'Illumina, alors que Louis-Philippe Lemieux Perreault s'est occupé de l'analyse des données provenant des micropuces d'Affymetrix. Richard Pearson a participé au design expérimental de l'étude, s'est occupé des analyses préliminaires des données provenant des micropuces d'Affymetrix et a révisé le manuscrit. Marie-Pierre Dubé a participé au design expérimental de l'étude et a révisé le manuscrit. Chris Spencer et Peter Donnelly ont supervisé de manière égale le projet décrit dans le manuscrit. Ils ont participé au design de l'étude et à sa coordination, ainsi qu'à la révision du manuscrit. Tous les auteurs ont lu et ont approuvé le manuscrit final.

To be submitted to PLoS Genetics

Methodological Challenges in Genome-wide Association Studies of Rare Copy Number Variants

Eleni Giannoulatou[1,†], Louis-Philippe Lemieux Perreault[1,2,†], Richard Pearson[1], Marie-Pierre Dubé[2], The Wellcome Trust Case Control Consortium 2, Chris Spencer[1,‡], Peter Donnelly[1,3,‡,*]

[1]Wellcome Trust Centre for Human Genetics, University of Oxford, Oxford, UK

[2]Montreal Heart Institute, Université de Montréal, Montréal, Canada

[3]Department of Statistics, University of Oxford, Oxford, UK

[†]These authors contributed equally to this work

[‡]These authors jointly supervised this work

7.1. Introduction

Copy number variation (CNV) is ubiquitous and responsible for much of the variation between human genomes. It has been suggested that CNVs might also account for important differences in genetic risks of disease susceptibility, that have not been previously discovered using SNPs and that would account for the remaining "missing" heritability [110]. Although common CNVs typed using existing array technologies were found unlikely to contribute greatly to the genetic basis of common human disease [99, 48], rare CNVs, either individually or in aggregate, have been suggested to be associated with a range of common diseases, especially neurodevelopmental diseases [154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 16, 165, 166, 167, 168].

The genome-wide association studies (GWAS) of rare CNVs typed using arrays face additional challenges compared to traditional SNP-based GWAS. Limitations due to genotyping errors, subject ascertainment and other experimental design issues such as sample size, population structure and sample relatedness are not specific to CNV studies and have been addressed extensively in the field of SNP-based GWAS [169, 170, 171]. However, in CNV-based studies, further factors can cause artefactual associations or limit the power to discover an association between a copy number variant and a phenotype.

The current association studies of rare CNVs are generally based on the *ab initio* discovery of variants across the genome of healthy and diseased individuals. The detection of these variants is limited by the signal-to-noise ratio, dynamic range and coverage of the array platform used, the reference sample or pooled samples as well as the statistical method employed. Using different technology platforms and data processing algorithms can produce CNV data that vary in quality and often have both high false-positive and false-negative rates [172]. It is now expected that no single approach can identify all types of copy number variation and the inferred length of known rare variants has been shown to be dependent on the technology used [173].

Most often, no distinction can be made on whether these variants are rare, inherited CNVs or occur *de novo*. The copy number variants detected in multiple individuals that originate from the same mutation event will rarely have the same breakpoints. Therefore, the identification of recurrent variants resulting from the same rare inherited polymorphism is not straightforward and can only be based on heuristic assumptions. Overlapping variants between individuals can also differ in comparison to an heterogeneous group of variants within the same large genomic region [174]. Unless family information is used, the distinction between inherited or *de novo* CNVs cannot be made easily. Since the rate of sporadic structural mutation might be lower than the rate of CNV inheritance, the occurrence of independent *de novo* CNVs at the same locus might be more significant and associated with sporadic genomic disorders [175]. However, all the variants discovered are treated the same way and their frequency calculation is dependent on the heuristic

rules employed to identify recurrent mutations.

The association studies of rare or *de novo* CNVs are prone to additional sources of bias, some of which have been addressed in previous studies. In order to achieve a low-false positive rate, a high false-negative rate is often allowed [175], introducing a selection bias when CNVs with frequency less than 1% in the population are only chosen for association testing. Additionally, although longer variants were initially assumed to be causative [176] they are now mostly selected for because of their higher confidence [154, 163, 164, 16, 165, 168, 177]. This also contributes in a selection bias when smaller variants, that can now be typed with similar accuracy, might also contribute to disease susceptibility. Finally, ascertainment bias is introduced when specific variants are investigated further in controls when only found in cases [175].

Previous studies have investigated the inconsistencies between platforms and methodologies when identifying CNVs in control populations [172, 178, 179, 180]. In addition, comparisons between existing calling algorithms have been performed extensively for different array platforms [181, 81, 182, 183]. In this study, we extend the emerging challenges of genome-wide association studies of rare CNVs using SNP arrays, that can range from the CNV detection and the association testing procedures to the interpretability of the findings.

7.2. Materials and Methods

7.2.1. Samples

345 technical replicates of the HapMap sample NA12878 were genotyped on Affymetrix's Genome-Wide Human SNP Array 6.0, while 68 replicates of the same sample were also genotyped on Illumina's custom Human660W-Quad (a WTCCC2 custom array comprising the Human550 and a set of circa 6,000 common CNVs from the Structural Variation Consortium [99, 48]). Samples and resulting CNV calls were filtered according to the criteria described in the Quality Control section. Structural variation data for the same sample, sequenced in high-coverage, were obtained

Table VII.I.: Breakdown of control samples analysed for CNV. Samples were filtered according the criteria described in the Quality Control section.

Cohort	Number of samples Affymetrix 6.0		Number of samples Illumina 1.2M		Common between samples	
	Before QC	After QC	Before QC	After QC	Before QC	AFter QC
NBS	2,987	2,512	2,731	2,412	2,715	2,131
58C	2,997	2,662	2,867	2,526	2,857	2,318
POBI	2,930	2,485	2,912	2,436	2,879	2,246

from the 1000 Genomes project [3]. We extracted long deletions (>100 kb) identified on NA12878 that passed the 1000 Genomes filtering criteria.

Population control DNA samples from three sources were genotyped on Affymetrix's Genome-Wide Human SNP Array 6.0 and Illumina's custom Human1.2M-Duo (a custom array designed by WTCCC2 comprising the Human1M-Duo and the common CNV content described above). The control samples consisted of ∼3,000 healthy blood donors recruited from the National Blood Service (NBS), ∼3,000 samples from the 1958 Birth Cohort (58C) obtained from Epstein-Barr virus-transformed cell lines from individuals born in England, Wales and Scotland during 1 week in 1958 and ∼3,000 samples from the Wellcome Trust-funded People of the British Isles (POBI) DNA collection, obtained from rural populations throughout the British Isles. A breakdown of the samples is given in Table VII.I.

7.2.2. Calling

Copy Number Variants can be detected within each sample using data from SNP arrays. The CNV calling algorithms are designed to take normalised intensity measurements from SNP array data and infer regions of increased or decreased copy number for each sample. The resulting CNV calls can vary depending on the choice of the platform and calling algorithm used.

The array products of the two platforms used differ in technology as well as in probe selection strategy. The Affymetrix SNP Array 6.0 features 1,842,285 probes (909,390 SNP probes and 932,895 monomorphic CN probes) with an average probe density of 6 probes per 10 kb. Less than half

of the SNP probes on the Affymetrix array are haplotype-tagging "tag SNPs" identified by the different phases of the International HapMap Consortium [184, 43], the rest are unbiased random SNPs chosen to cover the genome, excluding regions that cannot be monitored due to sequence restrictions of the assay. The majority of copy number probes (∼700,000) selected are also evenly spaced along the genome, while ∼200,000 probes specifically target ∼5,600 CNV regions from the Toronto Database of Genomic Variants. The Illumina semi-custom Human660W-Quad and Human1.2M-Duo arrays consist respectively of 660,447 (594,398 SNP probes and 66,049 CN probes) and 1,238,733 probes (1,158,605 SNP probes and 80,128 CN probes) with an average probe density of 2 and 4 probes per 10 kb respectively. The majority of Illumina's SNP probes are based on "tag SNPs" as well as on SNPs inside genes. The copy number probes were customly selected to target ∼6,000 common CNVs from the Structural Variation Consortium [99, 48].

With such a high resolution of markers found in the most recent arrays, many algorithms have been developed that allow SNP arrays to measure copy number changes across the genome. These methods take normalised intensity measurements obtained for the two alleles (X and Y) and infer the copy number state at each probe of a sample by taking into account that adjacent probes share the same copy number state. Recent studies have shown that the most accurate algorithms for calling CNVs are the ones that employ Hidden Markov Models to model the spatial dependence of probes and that have been specifically designed for data from a specific platform [178, 181]. In agreement with these findings, we applied Birdseye from the Birdsuite framework [65] and QuantiSNP [75] in order to call CNVs using data from the Affymetrix and Illumina SNP arrays, respectively.

QuantiSNP employs two measures for CNV estimation, provided by Illumina: the total fluorescent intensity signals from both alleles at each probe of the subject compared to expected intensity (referred to as the "log R Ratio": $LRR = log_2 = \frac{R_{subject}}{R_{expected}}$, where $R = X + Y$) and the relative ratio of the fluorescent signals between two alleles at each SNP calculated using the known B allele frequencies of the three canonical clusters (referred to as the "B Allele Frequency", as the standardised value of the allelic contrast $\theta = \frac{2}{\pi} \arctan\left(\frac{x}{y}\right)$, which is the polar angle between X and

Y normalised to a $[0, 1]$ range). By incorporating both of these measures in an Objective Bayes Hidden Markov Model, QuantiSNP provides probabilistic quantification of copy number changes along a chromosome. A Bayes Factor is assigned to each region of copy number variation detected. This provides a probability measure of the strength of evidence from the data for the presence of a copy number variant in a region versus the null hypothesis that there is no variant. Samples are analysed one at a time.

Birdsuite uses a clustering algorithm on the intensities (X and Y of the two alleles) to compute the number of copies for each probe, then performs the CNV segmentation by a Hidden Markov Model using only the observed number of copies. The Birdsuite framework distinguishes between the calling (or genotyping) of common (targeted) copy number variants (Canary) and the discovery of rare copy number variants (Birdseye). Since we are interested only on rare CNVs, the output of Birdseye is used for our analysis. Each copy number variant detected is assigned a LOD score indicating the relative probability of the variant versus normal copy number in the region. Samples are called on a plate-by-plate basis to account for batch effects, requiring though a minimum number of samples per plate (40 samples), in order for the clustering algorithm to perform accurately.

7.2.3. Quality Control

The quality of the SNP array data relies on many factors, including the quality and quantity of the input DNA and the efficiency of probe sequence hybridisation. Quality Control (QC) filtering is necessary in order to perform accurate SNP genotyping and CNV detection and calling. In addition, post-calling filters are subsequently applied to facilitate any downstream analysis. We have found that QC issues can vary according to the type of analysis performed, therefore the procedures that need to be followed for CNV analysis differ to those performed for SNP-based GWAS. Morever, although our aim was to be consistent and follow the same quality control procedures for both array types, the two technologies can exhibit slightly different artefacts that need to be accounted for.

7.2.3.1. Pre-calling exclusions

Affymetrix

Analysis of the raw data (CEL files) by a utility program provided by Affymetrix (*apt-geno-qc*), excluded noisy samples that might affect the clustering performance of Birdsuite.

Illumina

Illumina's proprietary software (BeadStudio) transforms the log R (total DNA intensity) and θ (allelic ratio or contrast) values into the appropriate standardised measures of Log R Ratio (LRR) and B allele Frequency (BAF), that QuantiSNP and other CNV detection algorithms employ. As described in [76], the log R Ratio compares the observed normalized intensity ($R_{subject}$) of the subject sample to the expected intensity ($R_{expected}$) based on the observed allelic ratio, $\theta_{subject}$, through a linear interpolation of three canonical clusters that correspond to genotypes AA, AB, and BB. These canonical genotype clusters are generated in advance by training on ∼120 normal (HapMap) samples for all HapMap probes, or using all the available samples in the cohort for the rest of the probes. In addition to computing $R_{expected}$, the observed allelic intensity ratio ($\theta_{subject}$) is used to estimate a quantitative B allele frequency for each probe in the given sample by using interpolation of the known B allele frequencies of the three canonical clusters (0, 0.5, and 1.0).

This standardisation provides a direct comparison to a reference, which is necessary for CNV detection. However, the canonical clusters generated for this procedure, can vary depending on the dataset used, particularly for the case of the custom probes that are not typed on other normal available "gold-standard" samples (HapMap). In order to account for the differences seen between the replicate samples of NA12878 that were genotyped together with different cohorts, we explored various probe exclusions before calling. In addition, the between-replicates comparison showed that QuantiSNP can be sensitive to these bad probes, resulting in fragmentation of long CNVs into smaller ones.

The pre-calling probe exclusions that were investigated included:

1. The union of all probes across replicates with missing information due to standardisation differences.
2. SNP probes with Frequency of Heterozygotes >0.55, Minor Allele Frequency < 0.05, HWE p-value $< 1 \times 10^{-10}$ and Missingness > 0.02.
3. Probes with high variability (Median Absolute Deviation of Log R Ratio > 0.25) that do not show high correlation with adjacent probes (average correlation with upstream and downstream probes < 0.5).

Although these exclusions showed an increase in sensitivity, the loss in specificity led to the inclusion of all the probes in our final dataset. For the unbiased comparison between the sample replicates, we removed samples with high number of missing probes by excluding the ones with less than 630,000 informative probes.

Finally, for the analysis of CNV-based control-control GWAS described in Sections 7.3.2 and 7.3.3, we excluded all probes that were missing (due to standardisation differences) in at least 1% of all samples (NBS, 58C and POBI) before calling.

7.2.3.2. Post-calling exclusions

For the CNV data found by both array platforms, we applied the same post-calling exclusions of samples and specific CNV calls. Samples with CNVs longer than 10 Mb were excluded as they could indicate cell-line artefacts. In addition, for each sample, in order to minimise false positives, we excluded CNV calls:

1. with confidence measure (log Bayes Factor or LOD score) below 10,
2. in non-autosomal chromosomes,
3. of length < 100 kb,
4. within $\pm$ 1 Mb of regions of known rearrangements such as Immunoglobulin genes or T-cell receptor genes (Supplementary Table S-X).

In Section 7.3.1, we analyse further the effect of different CNV exclusions that aim to increase sensitivity and the inevitable limitations of CNV detection. Finally, for the rare CNV-based GWAs

between the control groups, we excluded CNV calls found in more thatn 1% (or 5%) of the total population as described in the following section.

Samples that might contaminate any association results were also excluded based on the traditional SNP exclusion criteria described previously [185, 186, 187]. Hence, we excluded:

1. samples based on SNP call rate and heterozygosity,
2. related or duplicated samples,
3. samples whose gender did not match the supplier or whose gender is undetermined (based on the intensity of their X chromosome),
4. samples who were outliers when performing Principal Component Analysis to determine structure.
5. samples that failed a post-genotyping identity check using Sequenom,
6. samples with outlying chromosome intensity.

7.2.4. CNV frequency calculation

The analysis of rare CNV implies the need to distinguish recurrent events across a population in order to enable the calculation of CNV frequencies. This is a complicated task because of the low precision in detected CNV breakpoints that depend on the platform's probe density and signal-to-noise ratio. Heuristic rules have been applied in order to determine if an event is seen multiple times in a population or not. One approach is to create CNV regions (CNVRs) from artificially grouping CNV calls that partially or fully overlap [172, 46]. The new region will represent the union of overlapping CNVs (with the outermost boundaries of all events) and each CNV event that belongs to this region will be assigned the same CNV frequency estimated by the total CNV events contributing in the region. (See Box 1 at page 140 for illustrations.) Another approach for CNV frequency calculation does not involve any grouping of CNV calls. Each CNV from one sample is compared to the calls of all other samples in the population. Frequency is calculated by calculating the number of the CNV events that overlap between samples. The amount of overlap used between

CNVs can be any, for the CNVs to contribute to the total frequency, or above some threshold. The threshold commonly used is > 50% of reciprocal overlap between two CNV events (see Box 1, page 140).

We denote the set of all CNV events called (common and rare) in all our control samples (NBS, 58C, POBI) as $\mathbf{CNVE_{all}}$. After the creation of CNV regions for frequency calculation, the set of rare CNV calls for each frequency threshold used (1% and 5%) is denoted as $\mathbf{CNVE_{R\text{-}any}}$. Using the alternative "any overlap" rule with no prior CNV region creation, the set of rare CNVs is denoted as $\mathbf{CNVE_{any}}$. Finally, the set of rare CNVs created by using the "50% reciprocal overlap" rule is denoted as $\mathbf{CNVE_{50}}$. For examples of these rules, see Box 1, page 140.

7.2.5. Creation of CNV regions for association testing

In order to test for disease association of rare CNVs occuring at the same genomic locus, it is common to isolate the rare variants that affect the coding sequence of genes as well as their nearby conserved elements [175]. For this procedure the following approaches are followed:

1. For each genic region (from a gene's transcription start to end) we calculate the number of rare CNV events that affect it by partial or complete overlap in both cases and controls. Using the calculated frequencies, association testing is performed for each gene. The set of rare CNVs is created after the frequency exclusions described in the previous section.
2. Any overlapping rare CNV events in both cases and controls are merged into CNV regions (CNVRs). Each CNVR is assigned a frequency calculated by the proportion of samples that contribute calls into the region. All resulting CNVRs across the genome that span genes are tested for frequency differences at each gene between cases and controls. For the creation of these CNVRs, different rules are employed:
   a) CNVs with any overlap between them are merged into one CNVR
   b) CNVs with 50% reciprocal overlap between them are merged into one CNVR. An extension to this approach defines a CNVR as the minimum region containing 90% of the CNV calls of which it is comprised.

We applied the above rules to the set of rare CNVs we created prior to testing. CNV regions that were created using the "any overlap" definition on the $CNVE_{any}$ calls are denoted as $\mathbf{CNVR_{any}}$. CNV regions, also using the "any overlap rule" but on the $CNVE_{R\text{-}any}$ calls are denoted as $\mathbf{CNVR_{R\text{-}any}}$. CNV regions created with the "50% reciprocal overlap" rule on the $CNVE_{50}$ calls are denoted as $\mathbf{CNVR_{50}}$. In this case, every pair of underlying CNV calls in a given region satisfies the 50% reciprocal overlap rule. Finally, the CNV regions created with the "50% reciprocal overlap" rule with effective start and end positions defined as the positions where there is at least 90% of the events is denoted as $\mathbf{CNVR_{50.90}}$. In other words, the effective start and end positions are respectively the minimum and the maximum of the start and end positions of the underlying CNVEs, where there is at least 90% of the events. For examples of these definitions, see Box 1 at page 140.

7.2.6. Association testing procedures

We performed a post-quality control comparison between control cohorts, following the case/control design adapted by the majority of genome-wide analyses of rare CNVs [154, 155, 156, 157, 159, 160, 161, 162, 163, 164, 16, 165, 166, 167, 168, 177, 188, 189, 190, 191]. Three different data sets were created. These consisted of all the combinations of NBS, 58C and POBI cohorts, considering the samples of one cohort as cases and the samples of the other cohort as controls. All the steps (quality control, CNV frequency estimation and association testing) were then conducted independently on these three data sets ([58C, POBI], [NBS, 58C] and [NBS, POBI]). Applying the analysis pipelines described above, 6 sets of rare CNV events (and 8 sets of the corresponding CNV regions) for each cohort pair were created.

There are two main study designs for genome-wide association analysis of rare CNVs. The first one involves the examination of the aggregate frequency of all rare CNVs detected in cases in comparison to controls. This approach is often referred to as CNV burden (or CNV load) analysis. The aim of this analysis is to compare the amount of CNV events between cases and controls in terms of specific criteria. We employed most of the measures that have been used previously, which include the total number of rare CNVs, the mean size of the rare CNVs detected, the total kb

distance covered by rare CNVs or the number of genes intersected by one or more rare CNVs.

The comparison between cases and controls can be reported as the base two logarithm of the fold increase (or decrease), where a value of 0 represents no change, a positive value represents a higher number in cases and a negative value a higher number in controls. Additionally, a Generalised Linear Model-based CNV burden test is performed for the calculation of the Wald Z-statistic using logistic regression. Subsequently, 1,000,000 permutation steps were performed by shuffling the case and control labels to determine the significance of each burden analysis. A two-sided empirical p-value of the test was calculated as the proportion of sampled permutations where the absolute calculated Z-statistic was greater than or equal to the absolute observed Z-statistic.

The second study design for genome-wide association analysis of rare CNVs involves testing for disease association of rare CNVs occuring at the same genomic locus. To achieve this, we first calculated the number of CNVs (events or regions) that fully or partially overlap each known gene. To assess the significance in the difference between cohorts, the Fisher's exact test was used for each gene. Once again, 11,100 permutations were performed by shuffling the case and control labels in order to estimate the number of significant genes under the null hypothesis of no difference between cases and controls. All the sets of rare events and regions have been used for this gene-centric testing approach.

7.3. Results

The methodological challenges in performing genome-wide association studies of rare CNVs first stem from the limitations in CNV detection. In this work, we aim to address the accuracy of CNV detection by measuring the reproducibility between numerous replicates of one well-characterised sample. Subsequently, we target the secondary challenges of the CNV-based GWAS by focusing on the analysis design. Initially, we illustrate the disadvantages and limitations of a CNV burden analysis despite its potential power. Finally, we show how the *a priori* criteria determined for

testing for association between a phenotype and copy number variation at a specific locus, can affect the resulting significant findings.

7.3.1. Limitations of CNV detection using SNP arrays

Current approaches in CNV discovery involve searching into a single individual's genome for regions that show evidence of copy number variation. CNV detection algorithms are applied to the signal intensity SNP array data of each sample in order to discover CNVs and assign copy number states. These algorithms typically employ change-point methods [69] or Hidden Markov Models [65, 75] in order to take into account the spatial dependence of array probes that should exhibit the same copy number.

In order to minimise false positives, researchers tend to reduce the resulting discovery cohort by filtering CNV calls based on the following criteria: (1) *Size*: Short segments of copy number variation can be the result of noisy data, therefore they are regularly disregarded. The length cut-off used is more often 100 kb [158, 163, 16, 165, 168, 177], 500 kb [156, 159, 164] while recently a less stringent cut-off of 30 kb has been employed [162]. (2) *Number of probes*: Similarly, a high number of probes inside a detected CNV segment can indicate higher confidence. Thresholds such as 3 [165], 5 [162] 10 [154, 157, 158, 160, 159, 167] or 15 [164] number of probes are used. (3) *Confidence measure*: CNV detections algorithms often provide a confidence measure that indicates the reliability of a detected CNV within a sample. This can be a log Bayes Factor [75] or a LOD score [65]. Recommended thresholds are generally used; however depending on the data quality these measures are not always well-calibrated and more stringent thresholds are sometimes employed, despite a potential loss in sensitivityv [159, 177]. (4) *Other CNV exclusions*: In case-control experiments, regions that might give rise to artefactual associations are excluded. These often include CNVs residing in regions of known rearrangements (due to immunoglobulin genes or T-cell receptor genes) as well as in telomere or centromere proximal cytobands.

Measuring concordance between CNV calls is necessary in order to assess differences between algorithms, platforms, technical replicates as well as identify the recurrent CNV events across multiple individuals. The apparent factors that can affect concordance are the false positive and false negative rates in CNV detection. Another important factor is the differences in CNV breakpoints between detected CNVs that should correspond to the same mutation event, and the way these differences are handled. The rules that are generally used to determine if two recurrent CNVs correspond to the same event can be the following: (1) *Any overlap*: Any two CNVs in separate samples overlapping at any length are considered to be the same event [192, 193]. (2) *Overlap based on length*: Two CNVs in separate samples are considered to be the same event if their reciprocal overlap in length is above a specific threshold (such as more than 50% [156, 159, 167]). (3) *Probe-based overlap*: Two CNVs in separate samples are considered to be the same event if they overlap in the majority of their probes (such as more than 80% [177]).

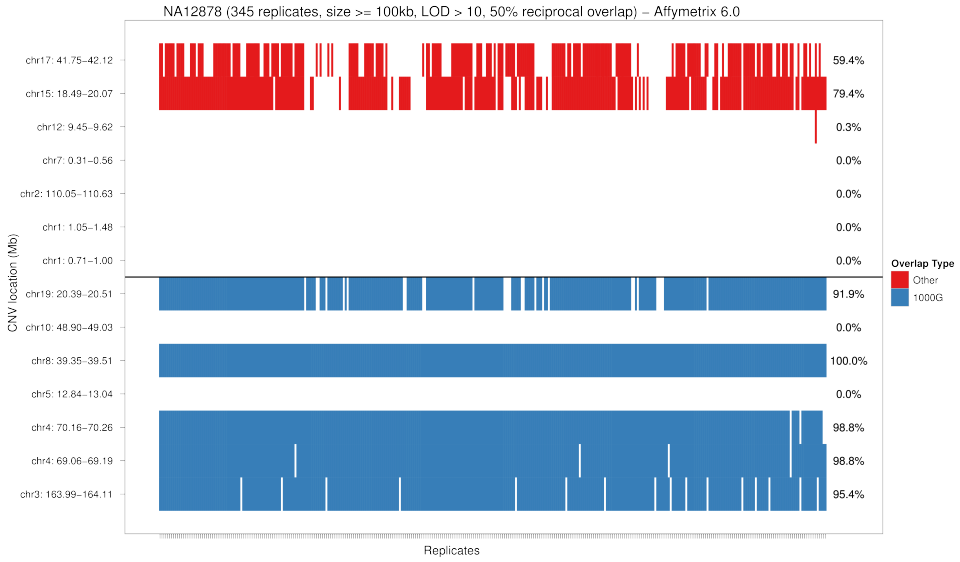
Different criteria or thresholds chosen can lead to substantially different CNV call rates, due to differences in sensitivity and specificity. Since there are no traditional quality control criteria applied in studies involving rare CNVs, large inconsistencies between studies are often seen [172, 173, 194, 80, 195]. The above choices are factors that underpin CNV detection and can affect any downstream analyses. Most importantly, the power to identify a disease association of a rare CNV is dependent on the accuracy of its detection.

We have performed an evaluation of concordance in CNV calling between replicates as well as in between platforms. One well characterised control sample (NA12878 from HapMap) has been typed multiple times on Affymetrix 6.0 (345 replicates) as well as on Illumina 660W (68 replicates). We applied Birdseye from the Birdsuite framework [65] and QuantiSNP [75] in order to call CNVs using data from the Affymetrix and Illumina SNP arrays, respectively. We assess the between-replicates reproducibility as well as their concordance to well-characterised data sets that are considered "gold standard" (obtained from the 1000 Genomes Project [3, 102]).

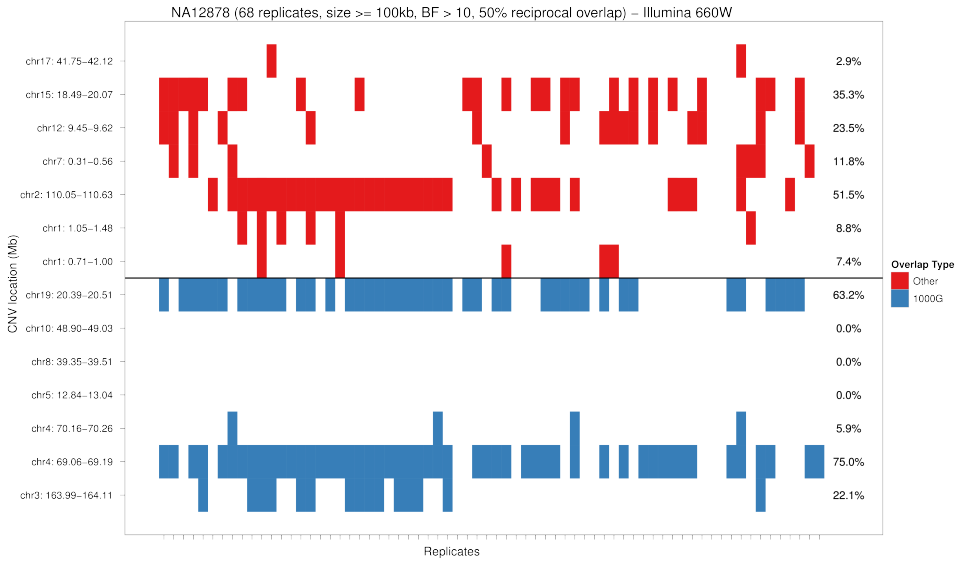
We extracted deletion genotypes identified on one individual (NA12878) sequenced in high-coverage (42×) that are longer than 100 kb. Out of the 23 deletions initially identified, 8 failed the 1000 Genomes Project filtering criteria (the most likely genotype was not 95% confident), 1 was an Immunoglobulin region and 3 regions had an allele frequency of 1, indicating that a deletion in the reference sequence was likely to be present (genotypes are estimated as homozygous reference, heterozygous and homozygous deleted). Moreover, two sets of 3 regions each were highly overlapping and thus were merged. The resulting set of deletions used consists of 6 heterozygous deletions and 1 homozygote deletion (chr4: 69.06-69.19 Mb) (Supplementary Table S-I). They were discovered in their majority using paired-end mapping and read-depth analysis and were all validated using either CGH (Comparative Genomic Hybridization Array) SAV (Superarray, an integration of available data from three array platforms Affymetrix 6.0, Illumina 1M, and a custom Nimblegen aCGH array with 4,938,838 probes into a high-density virtual array) or ASM (Assembly).

These "gold standard" calls were compared with the results of long (> 100 kb) deletions detected in multiple technical replicates of the sample HapMap sample NA12878 typed on the two array platforms, Affymetrix and Illumina. Varying rules were used for quality control as well as concordance assessment. The effect on sensitivity and specificity is investigated. Figures 7.1, 7.2 and 7.3 illustrate the between-replicates and between-platforms reproducibility in the detection of long deletions. For each of the 7 regions identified by the 1000 Genomes Project, the presence or absence of a call for each replicates is indicated. Moreover, 7 additional regions (> 100 kb), not present in the 1000 Genomes Project, but found in multiple replicates (more than 7% in both Affymetrix and Illumina) are shown.

Figure 7.1 illustrates the presence or absence of each deletion detected with the recommended confidence measure in each technical replicate of NA12878 typed on Affymetrix 6.0 and Illumina 660W. The 50% overlap rule has been applied to determine concordance between each 1000 Genomes Project region and the detected regions (shown in blue). In addition, the presence or absence of each of the 7 regions (> 100 kb) found in multiple replicates, using their outermost coordinates, is



(a) Affymetrix



(b) Illumina

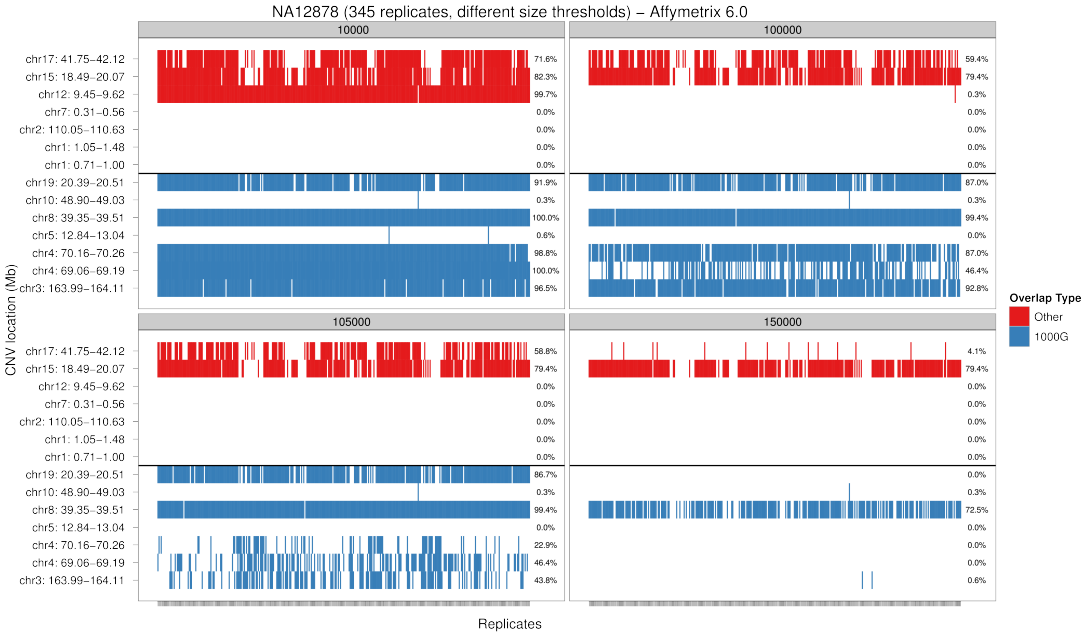
Figure 7.1.: Reproducibility of deletions detected in technical replicates of NA12878. The deletions detected in technical replicates of NA12878 using (7.1a) Affymetrix 6.0 or (7.1b) Illumina 660W are compared. The first 7 regions (red) are other deletions found by the two platforms (using their outermost coordinates), but not by the 1000 Genomes Project. The last 7 regions (blue) are deletions (> 100 kb) found by the 1000 Genomes Project. Numbers on the right or each row represent the percentage of replicates showing a deletion in the corresponding region. The overlap rule used for the 1000 Genomes regions is the 50% reciprocal overlap between the CNVs and the region, with a confidence threshold of more than 10 (LOD score or logBF). The rule used for the other regions is any overlap between the CNV and the corresponding region, with a confidence threshold of more than 10 (LOD score or logBF) and a minimum size of 100 kb.

indicated (shown in red). In this case, the 50% overlap rule is not applied since there is no "gold standard" region to compare these regions against. Affymetrix 6.0 succeeds in identifying 5 out of 7 "gold standard" deletions with a concordance varying between 91.9% and 100% of the replicates. On the other hand, Illumina 660W detects 4 out of 7 of the 1000 Genome deletions with a lower reproducibility varying between as low as 5.9% and as high as 75%. The additional 7 regions that have not been identified by the 1000 Genomes Project are presented here as potential false positives. Affymetrix 6.0 identifies two of these regions in 59.4% and 79.4% of the replicates, whereas Illumina 660W identifies all of these regions in much less of the replicates (concordance between 2.9% and 51.5%) supporting the hypotheses of potential false positives in these regions.

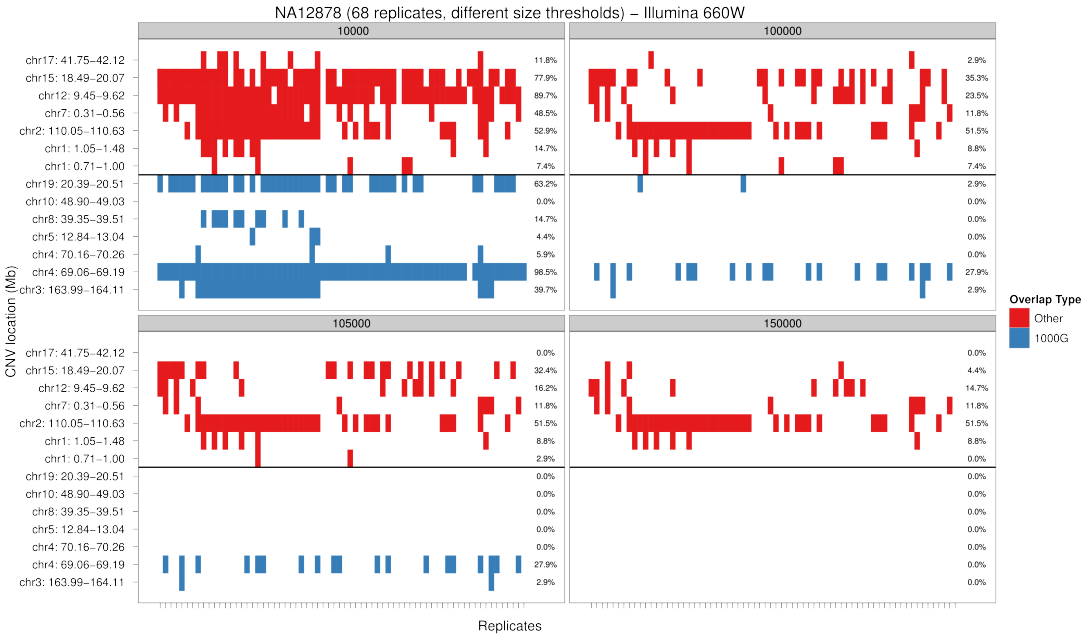
There are multiple reasons for the discordance between the two array platforms. The two arrays differ substantially in resolution and probe coverage as illustrated in Supplementary Figures S2 to S8 and Supplementary Figures S9 to S15 for the regions of interest. For example, in the case of the 3 deletions that are undetected by Illumina 660W in all of the replicates, the probe coverage is not sufficient while the inter-probe distance is too high for the CNV detection algorithm to identify any continuous variation in the regions (Supplementary Figures S12, S13 and S14). The difference in technology between the two platforms might also explain the level of discordance seen between the replicates. The signal-to-noise ratio varies between the two arrays, with Affymetrix showing a better separation between copy number states in the total DNA intensity signal (see Supplementary Figure S16). Given the intensity variation between replicates (Supplementray Figure S17), and the lower dynamic range of the Illumina platform, the resulting CNV calls can be substantially different between replicates. Additionally, the algorithm's sensitivity to outliers can create fragmentation of long CNVs into multiple smaller ones as seen in Supplementary Figures S9 to S15. Although the aim of this work is not to explore algorithmic differences in CNV detection, we should note that the discordance between the platforms can also be affected by the choice of the CNV detection method as it has been investigated in the past [181, 81, 182, 183].

To further investigate the low reproducibility seen between replicates typed on Illumina 660W, we explored the effects that could contribute to the variation in total DNA intensity. These include various batch effects as well as differences in the reference data employed for standardisation and calculation of the Log R Ratio and B Allele Frequency. All the replicates have been typed in different plates at different times, therefore we could not estimate the plate effect in the intensity variability. We did not observe a row or column effect in the total DNA intensity when we separated the replicates according to the rows and columns of the plates they were typed in (ANOVA p-values = 0.0849 and 0.1184 respectively, Supplementary Figure S18). The reference sample used for standardisation is either estimated using canonical clusters of a reference population (HapMap) or based on the intensity data of samples typed on the same cohort. Because of the variability between cohorts, we found that the between-replicates variance of the standardised total DNA intensity signal (Log R Ratio) is slightly higher compared to the total DNA intensity signal (Log R) especially for noisy low-intensity probes (see Supplementary Figures S19). However, no significant difference was detected as this standardisation is not modifying substantially the total DNA intensity variation between replicates. Therefore no apparent effect was discovered that could easily explain the discrepancy between replicate samples. The variability seen is most likely due to inherent technical variability that is not easily accounted for and which is critical to studies' reproducibility.

Figure 7.2 shows the concordance of the deletions detected in all the replicate samples, for varying size thresholds. The recommended confidence measure threshold is used for detection, while any overlap between calls is used to measure reproducibility. For a size threshold of 10 kb the concordance between the calls increases for both platforms and Illumina exhibits as high reproducibility as of 98.5%. This is caused by the tendency of the CNV detection algorithms to split a single CNV into more than one segment, due to variation in probe performance across the CNV. This effect can be confounded by genotype in that the homozygous state may lead to a single segment across the CNV, but the lower signal of the heterozygous state can lead to splitting. Thus multiple segments of different lengths can be obtained for a single long deletion. These would be discarded when a size threshold of 100 kb is used (Supplementary Figure S9 to S15). However, the size threshold



(a) Affymetrix



(b) Illumina

Figure 7.2.: Reproducibility of deletions detected using varying size thresholds in technical replicates of NA12878. The deletions detected in technical replicates of NA12878 for varying size thresholds using (7.2a) Affymetrix 6.0 or (7.2b) Illumina 660W are compared. The first 7 regions (red) are other deletions found by the two platforms (using their outermost coordinates), but not by the 1000 Genomes Project. The last 7 regions (blue) are deletions (> 100 kb) found by the 1000 Genomes Project. Numbers on the right or each row represent the percentage of replicates showing a deletion in the corresponding region. Each box represents a different size threshold (10 kb, 100 kb, 105 kb and 150 kb respectively).

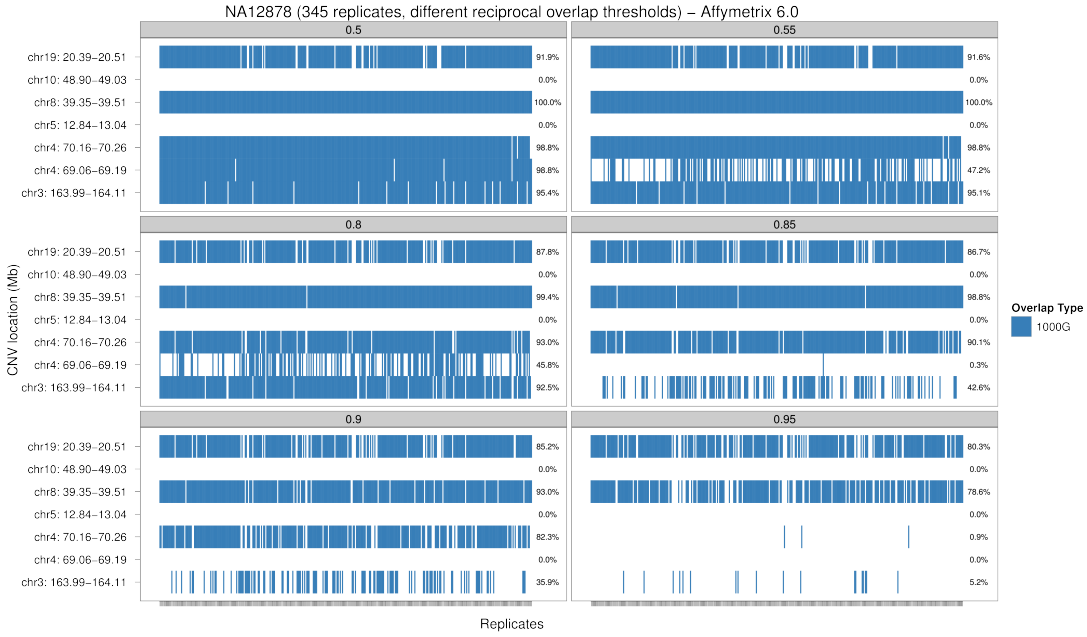
of 100 kb, despite a loss in specificity, is mainly used to avoid false positive calls. Figure 7.2 also illustrates that the reproducibility between replicates increases for the 7 additional regions not detected by the 1000 Genomes Project.

We should note that none of these potential false positive regions correspond to deletions longer 100 kb released by the 1000 Genomes Project that failed Quality Control. By examining the release set of the 1000 Genome Project including smaller variants, only one region (chr12: 9.45-9.62 Mb) shows some evidence of structural variation containing 35 small variants (average size 7400.429 bp) in region chr12: 9,524,311-9,620,930 Mb. Region chr15: 18.49-20.07 Mb is proximal to the centromere of chr15 which is a region prone for false positive variants detected using array data and was discovered as duplication in previous studies (according to the gold standard set collated by the 1000 Genomes Proect [3, 102]). In the set of smaller variants only one region (chr15: 19,790,211-19,794,130 Mb) of 3,919 bp size was shown to be deleted, whereas two other regions (chr15: 19,797,811-19,800,030 Mb and chr15: 19,866,911-19,888,930 Mb) did not pass Quality Control criteria (most likely genotype was not 95% confident). There was no evidence of structural variation spanning the region chr17: 41.75-42.12 Mb according to this release set. However since the genotypes that correspond to this region are homozygote reference, this might indicate that these deletions were not detected by the 1000 Genomes Project because of their presence in the reference sequence. It was detected in NA12878 in previous studies (gold standard set used for sensitivity assessment by the 1000 Genomes Project [3, 102]) using array data. According to this gold standard dataset collated by the 1000 Genomes Project for sensitive assessment, 12 regions were shown to be longer than 100 kb. 5 of these agree with 5 of the regions we used as gold standard with the exception of regions chr5: 12.84-13.04 Mb and chr10: 48.90-49.03 Mb. Of the remaining 7 regions only region chr17: 41.75-42.12 Mb was in our discovery set of long deletions.

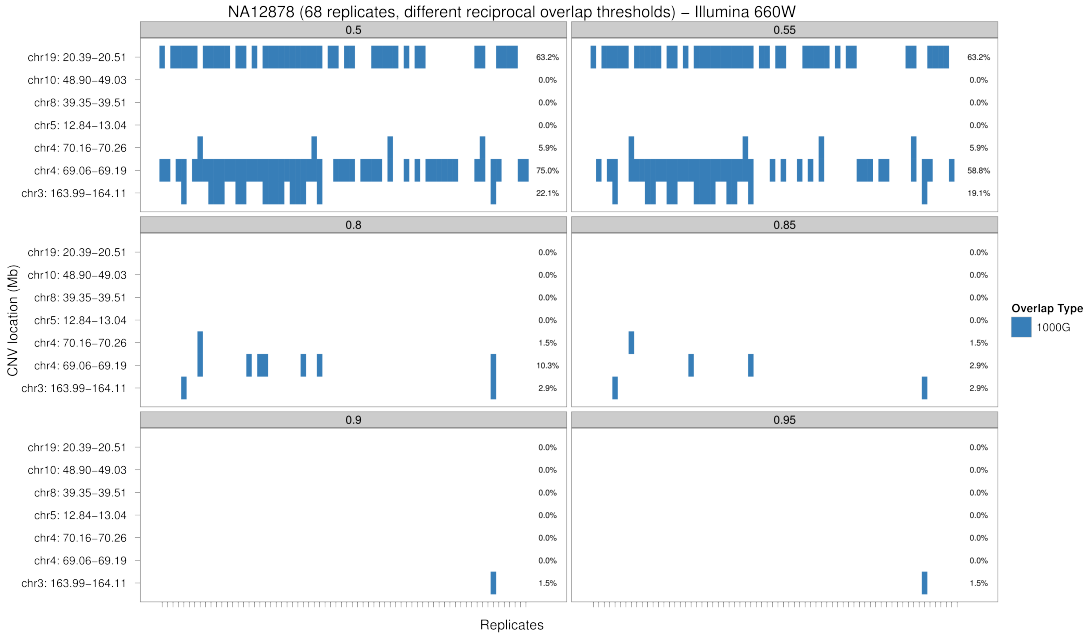
It is common practise in genome-wide association studies of rare CNVs to filter CNVs based on their length. However, excluding CNVs shorter than 100 kb restricts the analysis to a very small subset of the discovery set. To further investigate and quantify this, we fitted an exponential (or

geometric) distribution to the length of all CNVs discovered by Conrad *et al.* [48] as well as the set of all 1000 Genomes Project deletions using all HapMap data. This enabled us to estimate the probability of seeing CNVs or only deletions longer than 100 kb as 2.44×10^{-3} and 2.93×10^{-4}, respectively. Apart from the rarity of longer variants, this can furthermore be affected by the limitation of current CNV detection methods in accurately identifying long variants with precise breakpoints. Additionally, since such a low number of long CNVs is expected to be found in a sample, by applying a strict threshold, variants close to the cut-off value will be eliminated although could be potentially pathogenic. In this case, the association results would be highly distorted given the scarcity of these events.

Finally, Figure 7.3 illustrates how the concordance between replicates and the call accuracy varies for different thresholds of the deletions' reciprocal overlap. In this case, the concordance is estimated at the "gold standard" deletion sites. The potential false positive regions are not indicated here, since they are formed by the union of deletions found in the region, with no threshold on their overlap. It is expected that higher values of the reciprocal overlap threshold minimises the false positive calls. However, as it is obvious from Figure 7.3, higher values of this threshold results in a lower number of calls and lower reproducibility between replicates. This is due to the low accuracy in the estimation of the region breakpoints. When the start and end position of the detected deletions do not correlate perfectly with the actual breakpoints, a high overlap cut-off will result in a low call rate as deletions below the threshold will be discarded. For both platforms the concordance decreases dramatically and for an overlap threshold as high as 95%, only two deletion regions were detected using Affymetrix 6.0 with at least 78.6% concordance between replicates, whereas only one deletion region is detected in no more than one replicate on Illumina 660W. Therefore, the employment of stringent quality control measures, although it can reduce any false positive calls, might have large effects on the number of events that are used for downstream association analysis.



(a) Affymetrix



(b) Illumina

Figure 7.3.: Reproducibility of deletions detected using varying reciprocal overlap thresholds in technical replicates of NA12878. The deletions detected in technical replicates of NA12878 for varying reciprocal overlap thresholds using (7.3a) Affymetrix 6.0 or (7.3b) Illumina 660W are compared. The first 7 regions (blue) are deletions (> 100 kb) found by the 1000 Genomes Project. The last 7 regions (red) are other deletions found by the two platforms (using their outermost coordinates), but not by the 1000 Genomes Project. Numbers on the right or each row represent the percentage of replicates showing a deletion in the corresponding region. Each box represents a different reciprocal overlap threshold (50%, 55%, 80%, 85%, 90% and 95% respectively).

7.3.2. Rare CNV burden analysis

Previous studies on complex phenotypes such as schizophrenia [154, 167, 168], obesity [159], autism [162], bipolar disorder [163], attention deficit hyperactivity disorder [164] and heterotaxy [189] have found an increase in genome-wide CNV burden in cases in comparison to control samples, while others were unable to replicate some of these findings [158, 160, 196]. The CNV burden (or "CNV load") can be assessed by different measures such as, the number of rare CNVs per sample, the mean CNV size per sample, the total genome length covered by rare CNVs or the number of genes intersected by rare CNVs. This analysis is based on the assumption that the mechanisms and the resulting patterns of rare variation that is non-pathogenic are similar in cases and controls and thus an enrichment of pathogenic rare variants in cases would result in increase of rare CNV burden. The different criteria used are based on this assumption; by specifically considering the mean size of rare CNVs and the total kb extend of rare CNVs, the aim is to overcome the limitations of CNV detection [162, 168, 177]. In addition, by examining the number of genes intersected by CNVs, the effect of genic CNVs is directly investigated based on the assumption that causal variants act by the disruption of genes [162, 177]. Unfortunately, such an increase in rare CNV burden might not be directly linked to an actual change in expression levels, and further analyzes are required in order to assess the impact of such an increase in burden of rare CNVs.

In addition to the challenges of CNV detection, the frequency definitions that are used in order to estimate the set of rare CNVs (with frequency less than 5% or 1%) can vary according to whether it is assumed that these recurrent but rare CNVs arise from the same mutational event or not. CNV events that originate from the same mutational event, which has been segregating in a population, will rarely obtain the same breakpoints. In order to identify which of these events are rare, their frequency must be accurately estimated. However, researchers have been following various approaches, and no universal method exists. In a idealised situation, the CNV coordinates between recurrent CNVs originating from the same mutational event, will not differ dramatically, and merging individual CNV events into CNV regions would give satisfactory results. The definition of CNV region (CNVR) stems from the interpretation of CNV concordance based on any overlap

between events, where any two CNVs in separate samples overlapping at any length are considered to be the same event [172, 46]. Therefore, the merged CNV region will contain the union of the overlapping CNVs with the outermost boundaries of all events. The CNV frequency of each event is then defined as the proportion of samples that contribute calls to the CNV region. Depending on the data quality, such an approach might not work in practise since long false positive events can alter these coordinates significantly affecting any downstream analyses (see Box 1 at page 140). Under the assumption that CNV events could also occur sporadically, the construction of CNV regions is not necessary. In this case, the events are treated separately and their frequency is calculated either by calculating the frequency of their spanning probes (as followed by PLINK [128]) or by counting the proportion of overlapping CNVs across samples. Thresholds on the reciprocal overlap on the CNV lengths can be used (such as 50% reciprocal overlap [154, 162, 168, 197], see Box 1 at page 140), This approach is more widely used for estimating frequencies of CNV events necessary for CNV burden analysis as well as for identifying specific CNV loci enriched in cases in comparison to controls.

Rare CNVs are most often considered the ones found in less than 1%, or less often 5% in the total population consisted of both the cases and control cohorts. Some studies choose to identify rare CNVs using the frequency of recurrent events only in the control population given the assumed enrichment of these in the cases [190, 197]. However, such an approach biases the selection of rare variants in favour of those that have higher frequencies in cases and, as demonstrated previously for the case of rare SNPs, it could lead to erroneous conclusions [198].

Since there are many potential ways to group a collection of CNV events and different ways to handle the limitations of CNV detection, testing CNVs for disease association is not as constrained as in the case of SNP-based GWAs. Under the assumption that multiple, independent rare CNVs might contribute to disease risk, a joint testing of all rare events will provide more power than individually testing each rare variant. Therefore, the CNV burden analysis is potentially more advantageous that a genome-wide screening of each recurrent CNV. However, such an analysis

could be largely affected by the methodological challenges involved.

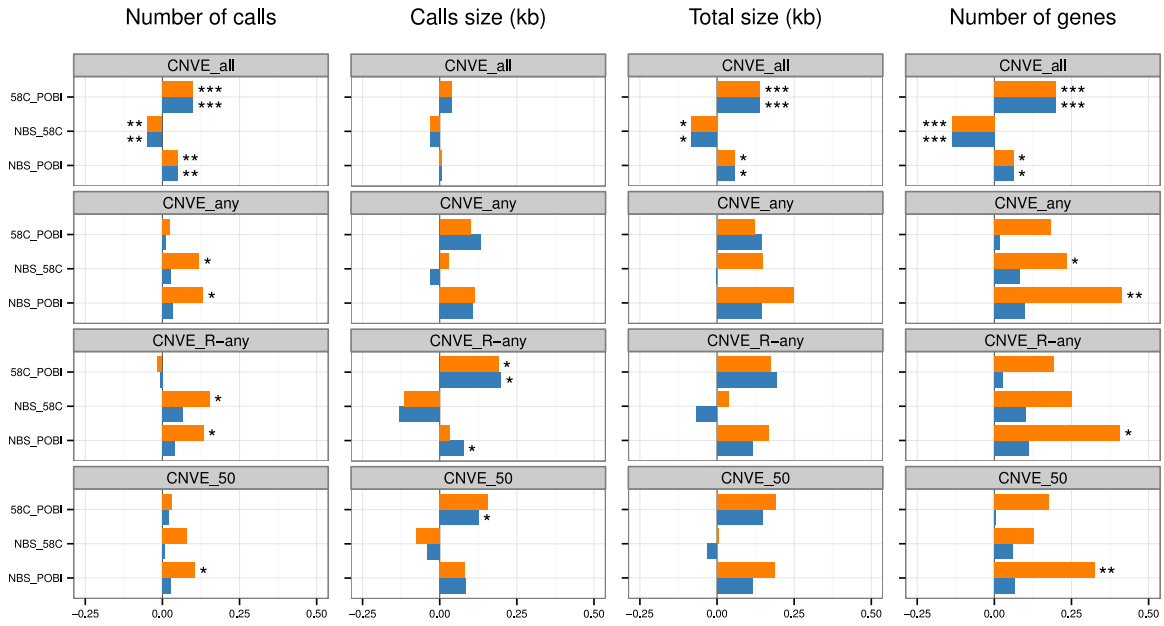
We have assessed whether the impact of rare CNVs on the risk for different disease phenotypes in terms of an individual's genome-wide burden can be confounded by the limitations of CNV detection. We have performed CNV burden analysis, in terms of the number of CNV calls per sample (including all or only rare variants longer than 100 kb), the estimated CNV size, the total length of the genome covered by CNVs and the number of genes intersected by CNVs between three control groups (∼3,000 NBS, ∼3,000 58C and ∼3,000 POBI) typed on both Affymetrix 6.0 and Illumina 1.2M, and by employing different approaches for CNV frequency estimation. Quality control filters were applied as described in the Methods section. In brief, related and noisy samples were removed, as well as CNV calls intersecting (or proximal to) immunoglobulin regions. The samples used for further analysis do not have any ancestral differences and have been tested for plate effects. Additionally, no differences in the median absolute deviation (MAD) distribution of the total DNA intensity between groups were present. We therefore expected to find no difference in the CNV burden between the control groups.

Figures 7.4 and 7.5 show the results of the CNV burden analysis we performed using the three control cohorts, NBS, 58C and POBI, typed in two arrays, Affymetrix 6.0 and Illumina 1.2M. The control comparisons were performed in pairs comprising three independent analyses of 58C vs. POBI, NBS vs. 58C and NBS vs. POBI. For each analysis, the set of CNVs tested were: (1) all CNVs - by assuming that the amount of common CNVs is similar between cases and controls, any enrichment detected in one of the two groups would be attributed to the rare CNVs (the set is denoted as $\mathbf{CNVE_{all}}$), (2) the set of rare CNVs determined by calculating the proportion of CNV events exhibiting any overlap between them for the frequency estimation (denoted as $\mathbf{CNVE_{any}}$), (3) the rare CNVs determined after the creation of CNV regions for the frequency estimation (denoted as $\mathbf{CNVE_{R\text{-}any}}$) and (4), the set of rare CNVs determined by calculating the number of CNV events that overlap reciprocally by more than 50% of their length for the frequency estimation (denoted as $\mathbf{CNVE_{50}}$). For examples of these definitions, see Box 1 at page 140. For all the rare

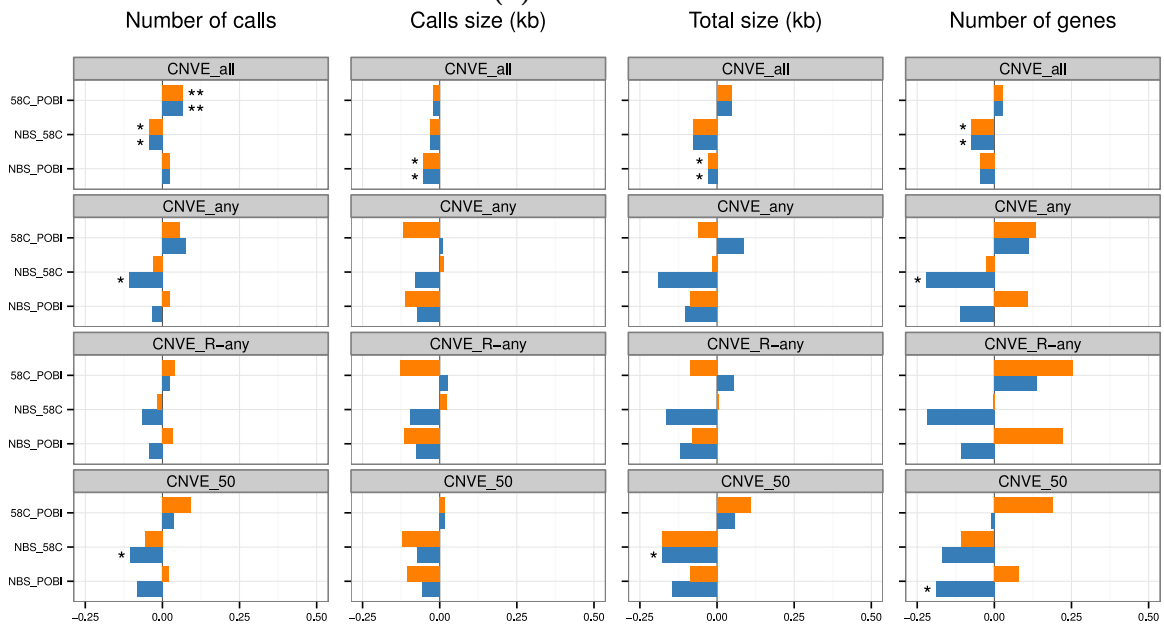
CNV calls, we employed two frequency thresholds of 1% and 5% of the total population consisted of two control groups at each time. The figures show the log fold change of (1) the number of calls per sample, (2) the mean CNV size per sample, (3) the total genome distance covered by CNVs and (4) the number of genes intersected by CNVs per sample. A value of zero indicates no difference in the fold change between the two cohorts as it is expected a priori. 1,000,000 permutation steps were performed by shuffling the case and control labels to determine the significance of each burden analysis. The analyses were performed twice including all CNVs or only deletions. Supplemental Tables S-II to S-IX include the fold change values for each type of analysis.

There is no clear consistency in the CNV burden results between the two platforms used, the different measures used to assess the CNV burden, the different pipelines employed for the determination of the set of (rare) CNVs, the two frequency cutoffs or the CNV types (all or deletions only) investigated. The analysis results using the calls detected by the Affymetrix 6.0 (Figure 7.4), show few significant differences between the two groups, although these were not uniform for different burden criteria. For example, although the number of genes intersected by rare CNVs ($< 1\%$ frequency, using 50% overlap rule) seems to be higher in the NBS samples in comparison to the POBI samples (p-value $< 1 \times 10^{-3}$), the total distance covered by CNVs or the average CNV size in NBS samples do not show any statistically significant difference. On the other hand, numerous significant differences between control groups were found when typed on Illumina 1.2M (Figure 7.5) consistently across the CNV burden indicators but not in agreement with the findings of CNVs detected by Affymetrix 6.0. This indicates that these findings are the results of artefacts caused by the CNV detection weaknesses of the Illumina platform. For both platforms, the number of significant differences decreases when only the deletions are investigated. This is expected since the duplications are more prone to be false positive events affecting the burden analysis.

No difference in CNV load is expected to be found between these groups. Stringent QC has been applied but the methodological variability that exists can skew the results significantly. This might explain the failure of previous studies to replicate the CNV burden excess reported in some phe-

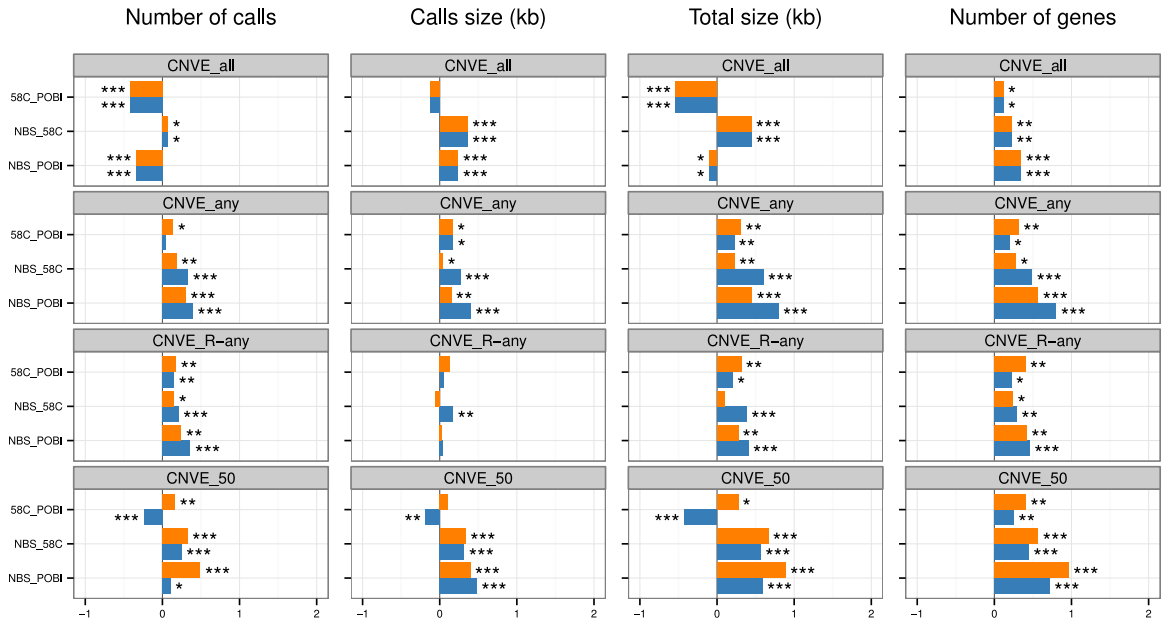


(a) All CNVs

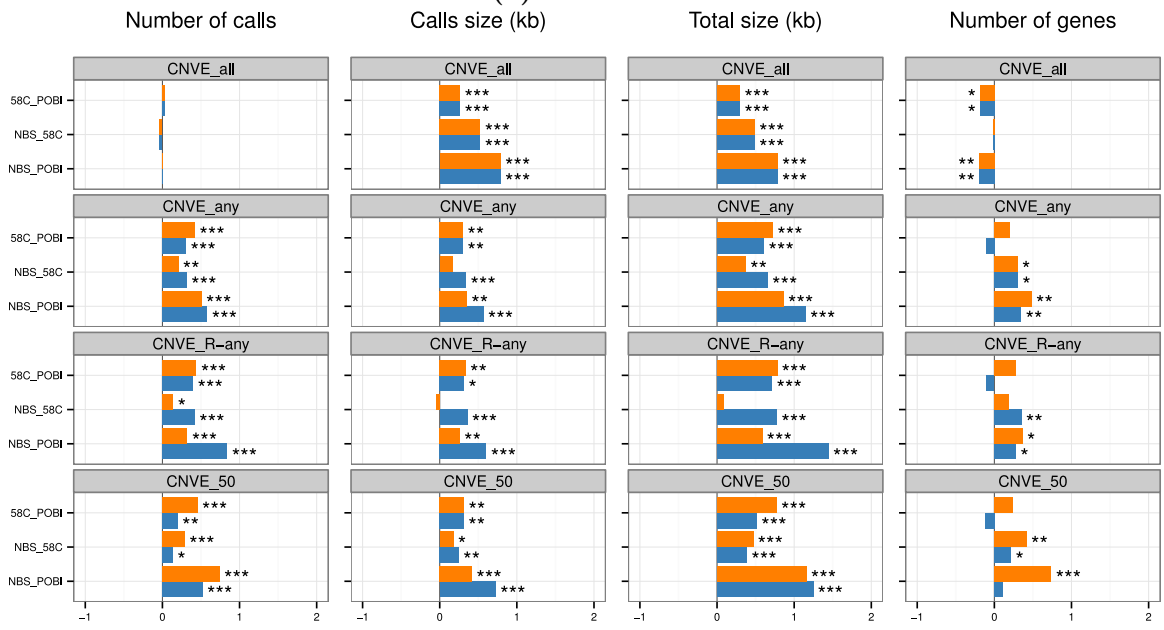


(b) Only deletions

Figure 7.4.: CNV burden analysis results for control groups typed on Affymetrix 6.0. The $\log_2$ fold change between control groups for the total number of calls, the mean size of calls (kb), the total extent of calls (kb) and the number of intersect genes using (7.4a) all CNVs or (7.4b) only deletions detected using Affymetric 6.0. Each box represents a different frequency definition used to determine the set of (rare) CNVs (CNVE_{all}, CNVE, CNVE_{any}, and CNVE_{50}, respectively [see main text for more information]). The two frequency thresholds used are shown ($< 1\%$ in orange and $< 5\%$ in blue). Note that the range of $\log_2$ fold change shown here is different from the range shown in Figure 7.5. The significance cutoff values are *: < 0.05, **: $< 1 \times 10^{-3}$ and ***: $< 1 \times 10^{-6}$, calculated using one million permutations.



(a) All CNVs



(b) Only deletions

Figure 7.5.: CNV burden analysis results for control groups typed on Illumina 1.2M. The $\log_2$ fold change between control groups for the total number of calls, the mean size of calls (kb), the total extent of calls (kb) and the number of intersect genes using (7.5a) all CNVs or (7.5b) only deletions detected using Illumina 1.2M. Each box represents a different frequency definition used to determine the set of (rare) CNVs (CNVE_{all}, CNVE, CNVE_{any}, and CNVE_{50}, respectively [see main text for more information]). The two frequency thresholds used are shown ($< 1\%$ in orange and $< 5\%$ in blue). Note that the range of $\log_2$ fold change shown here is different from the range shown in Figure 7.4. The significance cutoff values are *: < 0.05, **: $< 1 \times 10^{-3}$ and ***: $< 1 \times 10^{-6}$, calculated using one million permutations.

notypes. Although ideally the joint testing of all CNVs could be more powerful than testing each CNV individually, it is very sensitive to experimental errors and technical artefacts. This collective testing of rare CNVs seems to aggregate the confounding effects that exist due to data quality, calling sensitivity and identification strategy of recurrent CNV events, at multiple sites.

Another disadvantage of such an analysis is the lack of interpretability of the potential findings. Specific factors are not easily identified and the contribution of individual CNVs and their link to specific genes is not straightforward from such an analysis. This is further limited since no a priori distinction is made between rare inherited and *de novo* CNVs.

7.3.3. Testing for Associations

An alternative strategy to CNV burden analysis is to assess whether specific loci exhibit an excess of CNVs in cases compared to controls. This analysis involves testing for disease association of rare CNVs occurring at the same genomic locus. The aim is to identify these CNVs and potentially the mechanism by which they confer disease risk. The events tested occur with low frequency ($< 1\%$ or 5% of the total cases and control samples) and it is assumed that if pathogenic they would occur more frequently in cases than in controls.

Previous studies have followed several strategies to identify pathogenic CNVs. In addition to the challenges of CNV detection and frequency calculation, this analysis can be highly sensitive to the determination of the regions for testing. In contrast to the pre-specified variants used in SNP-based GWAS, the detected CNV events will rarely be the same between cases and controls exhibiting again different breakpoints and length. Since this has been proved difficult, some studies choose to only test the overlapping rare CNVs that are found in cases but not in any control samples, as that would support their causality [16, 197, 199]. However, such an approach is subject to ascertainment bias [175].

Alternatively, in order to identify CNVs that confer risk to diseases the following approaches have been employed. (1) *CNV-based approach*: Each CNV event is first assigned a frequency using one of the methods described in the previous section. After applying some heuristic rules to identify the same events between cases and controls (*e.g.* $> 50\%$ reciprocal overlap), association testing is performed for each single event. (2) *CNVR-based approach*: CNV events identified in both cases and controls are merged into unique CNVRs. Testing is performed for each CNVR using the frequency calculated by the CNV events that contribute to the CNVR for cases and controls separately. As we showed in the previous section various rules can be followed for the creation of CNV regions (using any overlap between events, more than 50% overlap and using the coordinates of more than 90% of the events in the region). (3) S*egment-based approach*: In this analysis the total number of events within a sliding window (*e.g.* $\pm$ 50 kb) is compared between cases and controls. The testing results would indicate a segment (*e.g.* of 100 kb) that might exhibit higher frequency of CNVs in cases compared to controls (PLINK [128]) (4) *Probe-based approach*: The total number of events overlapping each single probe are compared between cases and controls. A scoring approach is applied that scans the genome for consecutive probes with more frequent copy number changes in cases compared to controls defining significant CNV regions a posteriori [158, 161, 128] (PLINK) and (5) *Gene-based approach*: The total number of CNVs intersecting each known gene is compared between cases and controls [161, 162, 177, 190, 128] (PLINK). The number of CNVs that fully or partially overlap each is first calculated and subsequently, association testing is performed for each gene.

The gene-based approach has the advantage that CNVs with different breakpoints affecting the same gene are pooled into units with the same assumed functional impact, which provides more power than testing each variant individually [175, 190]. This approach can directly pinpoint a gene as being relevant to the trait, rather than investigating the functional role of any intergenic region that might be found statistically significant. However, false associations can occur for genes lying in breakpoints of rare CNV events, caused by the variation in boundary determination of CNV calling. To minimize this problem, CNV events can be first merged into CNV regions using appropriate rules, before the calculation of variants (CNVRs) that disrupt each gene in cases and controls (see

Box 1, page 140). Moreover, the approach of creating CNVRs can be significantly improved when the merging of CNVs employs the approach of CNV concordance based on CNV length. Two CNVs in separate samples can be considered to be the same event and therefore merged into the same CNVR only if their reciprocal overlap in length is above a specific threshold (*e.g.* 50% [156, 159, 165]).

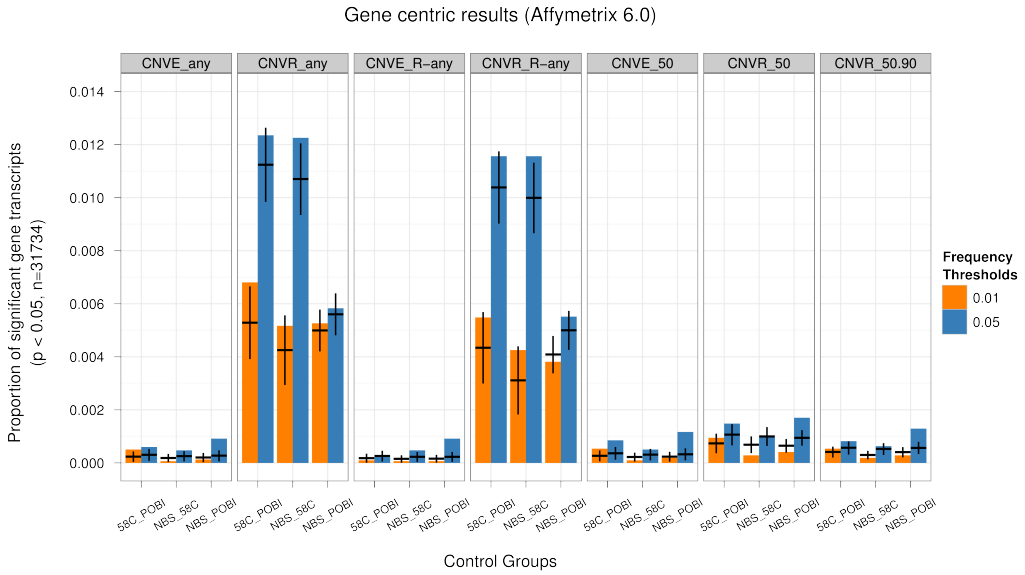
We should note that all the strategies for identifying disease association of CNVs would either perform testing for all types of events with deletions and duplications lumped [154, 159, 164, 177], separated by type [158, 161, 168, 189, 191, 196] or both [190]. Given the rarity of these events, few CNVs are expected to be found in both allele types and significant findings are often separated by type although a lumped analysis has been performed [154, 159]. Occasionally, it is not clear if the genome-wide screening has initially been performed separately for the two CNV types or not [16]. This is important for replication studies, as the lumped analysis can alter the resulting association results when only an increase or decrease of the gene-dosage affects the disease risk. Since deletions are shown to be more accurately typed than duplications and have more often been founded to be associated with disease [160, 163], frequently studies target specific deletion regions for association analysis [156, 188].

The genome-wide association screening of rare pathogenic CNVs suffers from the lack of analysis constraints resulting in many potential methodological strategies. The strategic choice for the calculation of CNV frequency, the determination of the interrogated regions, the stratification by type or other characteristics (*e.g.* genic CNVs) in addition to the limitations of the detected CNVs could potentially affect the resulting findings by either causing artefactual associations or by limiting the power to discover an association between a copy number variant and a phenotype. We have assessed whether such choices can alter the association analysis results by performing a genome-wide screening of rare CNVs between the three control cohorts (NBS, 58C, POBI) typed in two platforms (Affymetrix 6.0, Illumina 1.2M) following the case control design adapted by the majority of rare CNV-based GWAS [154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 16, 165, 166, 167, 168, 177, 188, 189, 190, 191, 196]. Given that this analysis is a post-quality control

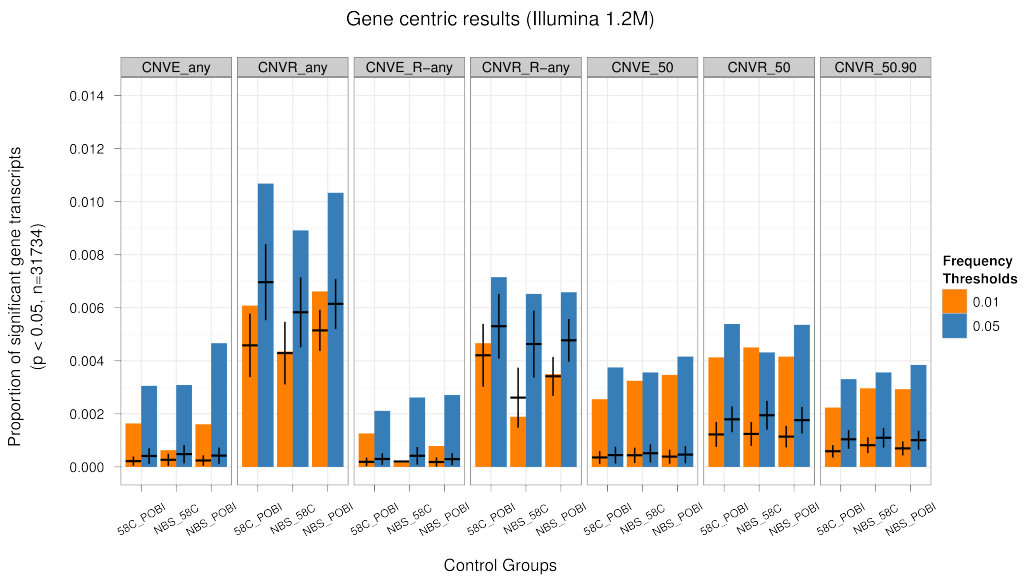
comparison, we expect to find no loci exhibiting significant differences between the groups, other than the number that it is expected by chance. A diagram of the analysis pipeline is shown in Supplementary Figure S1.

We focus on long (> 100 kb) deletions that are found in less than 1% or 5% of the total population of the two cohorts compared. Figure 7.6 presents the resulting number of significant gene transcripts for each methodology used. We employed several of the a priori criteria used to determine the regions for testing, focusing on a gene-based approach. For each genic region (from a gene's transcription start to end) we compared the number of rare CNV events that affect it by partial or complete overlap in both cases and controls. In addition, we repeated this analysis by first merging any overlapping rare CNV events in both cases and controls into CNV regions (CNVRs). All resulting CNVRs across the genome that span genes were tested for frequency differences at each gene between cases and controls. For the creation of these CNVRs we used the "any overlap" rule, the "50% reciprocal overlap" rule as well an extension to this approach by defining a CNVR as the minimum region containing 90% of the CNV calls of which it is comprised. We applied these rules to the set of rare CNVs we created prior to testing. CNV regions that were created using the "any overlap" definition on the CNVE_{any} calls are denoted as $\mathbf{CNVR_{any}}$. CNV regions, also using the "any overlap rule" but on the $\text{CNVE}_{\text{R-any}}$ calls are denoted as $\mathbf{CNVR_{R\text{-}any}}$. CNV regions created with the "50% reciprocal overlap" rule on the CNVE_{50} calls are denoted as $\mathbf{CNVR_{50}}$. In this case, every pair of underlying CNV calls in a given region satisfies the 50% reciprocal overlap rule. Finally, the CNV regions created with the "50% reciprocal overlap" rule with effective start and end positions defined as the positions where there is at least 90% of the events is denoted as $\mathbf{CNVR_{50.90}}$. See Box 1 at page 140 for more information.

To assess the significance in the difference between cohorts, the Fisher's exact test was used for each gene transcript. The number of recurrent CNVs that need to be observed at the same locus for the finding to be significant might not be the same for different studies and alternative analysis methodologies. Such a number depends on parameters such as the rate of *de novo* variants (as well



(a) Affymetrix



(b) Illumina

Figure 7.6.: Proportion of gene transcripts found statistically significant between control cohorts typed in both Affymetrix and Illumina. Proportion of significant gene transcripts ($p < 0.05$) found for each analysis and for each frequency or CNVR definition (CNVE, $CNVE_{any}$, $CNVE_{50}$, $CNVR_{any}$, $CNVR_{50}$, and $CNVR_{50.90}$, see Box 1 for definitions) using only deletions > 100 kb from (7.6a) Affymetrix 6.0 or (7.6b) Illumina 1.2M. Both frequencies thresholds used for rare variants are shown ($< 1\%$ in orange and $< 5\%$ in blue). The error bars indicate the range of the number of gene transcripts expected to be found by chance as calculated by 5,800 permutations.

as the rate of rare CNV inheritance) and the number of locations in the genome that these CNVs occur [200]. However, these parameters can not be estimated easily as they are highly affected by the false positive and false negative rates. It is critical that most studies do not distinguish between rare inherited CNVs and *de novo* mutations although that would also provide additional value for the association testing. The probability of observing even a small number of *de novo* mutations in the same locus would be more unlikely than that of rare variant that has been segregating in a population with a specific, despite low, frequency. To estimate the number of significant gene transcripts under the null hypothesis of no difference between cohorts, 11,100 permutations were performed by shuffling the case and control labels. The error bars in Figure 7.6 present the range of number of gene transcripts expected by chance for each methodological strategy followed.

The use of events without the construction of regions for the testing produces less false associations. However, these can still occur either for genes lying in breakpoints of rare events (Supplementary Figure S20) or for gene lying between events that have been split by the detection algorithm (Supplementary Figure S21). The CNVR-based analysis for the frequency estimation tend to find higher number of significant gene transcripts between cohorts. This is expected since merged CNVRs usually span larger regions that can contain higher numbers of gene transcripts and although permutations performed account for this difference, they tend to exhibit higher numbers that we would expect by chance. Supplementary Figures S22 and S23 present examples of artefacts that are introduced when such analyses are performed. Merged events affect the frequency of the region resulting in spurious association results ($CNVR_{any}$, $CNVR_{R\text{-}any}$). We should note that these numbers decrease substantially when stricter rules are applied for the estimations of CNVRs ($CNVR_{50}$ and $CNVR_{50.90}$). In this case, different artefacts are introduced when for example the longest events in a region are the only ones overlapping a gene (Supplementary Figure S24) or when the 90% threshold is not met, excluding CNVs that overlap a gene from only one cohort and therefore yielding significant results (Supplementary Figure S25).

Similar to genotyping errors in SNP-based GWAS, CNV calling errors can produce spurious association results caused by uncertainty in breakpoints. Moreover, the way CNVs are clustered together into events or regions to identify recurrent variation is also responsible for these erroneous associations. In SNP-based GWAS, various quality control criteria are used that eliminate the occurrence of erroneously called SNPs. However, manual inspection of cluster plots is often required to confirm that associations are not due to these errors. In rare CNV-based GWAS no universal method exist that aims to eliminate the presence of these erroneous regions and therefore the inspection of plots such the Supplementary Figures S20 to 25 is required.

Figure 7.6 also shows that the number of artefactual differences is higher for the samples typed in Illumina 1.2M compared to the ones typed in Affymetrix 6.0. This was expected given the lower accuracy the variation is measured for these samples. However, the striking finding was that although the artefacts can be minimised for some of the methodologies (e.g. using $CNVE_5$, $CNVR_{50}$ and $CNVR_{50.90}$) for the variants discovered using Affymetrix, in Illumina calls the numbers of significant gene transcripts always remain higher than the numbers expected by chance. This indicates that such an analysis between cases and controls is impossible to perform without the presence of high number of false positive associations.

Finally, we should note that, for both platforms, the number of significant findings is usually much higher for the variants found with frequency less than 5% in the population than expected by chance. Since these variants have higher frequency, they would include more CNVs with different deletion alleles (one vs two copies deleted). For example, the occurence of homozygote deletions called in NBS samples is increased from 0.3% (4 out of 1339 total deletion calls) for the 1% frequency threshold to 0.59% (13 out of 2192 total deletion calls) for the 5% frequency threshold. Therefore more loci are now affected by both deletion types and by collapsing these types together the comparison can produce erroneous differences. This is important since each deletion allele can be tolerated differently and have a different effect in disease risk. By failing to capture the ranges of copy number loss, spurious associations can arise. Since these levels of copy number are not distinguished

in this type of analysis, the true disease association of these variants would not be easily determined.

The current association testing procedures applied to detect rare pathogenic CNVs may additionally suffer from interpretation difficulties. CNVs that may be found to have higher frequency in cases at a specific locus might span multiple genes that will exhibit the same level of significance. Proposing a mechanism behind the pathogenicity of these events might naively implicate a gene pathway or a set of genes when only a smaller number of these genes might be pathogenic. Additionally, long genes, which include brain genes, might be intersected by multiple different CNV events including some that might appear with similar rates in cases and controls. These issues have been addressed before when applied to pathway analysis in order to compare the rate of CNVs impacting specific gene sets in cases versus controls [201].

Finally, SNP information, although not used in this type of analysis, may be necessary not only to support the causality of a potential CNV but mainly to identify whether this variant is present on the same haplotype among all the cases that it occurs. This is important as it would indicate that the variant is not *de novo* but inherited rare variation with an appreciable frequency in the population. An integrated SNP and CNVs analysis would be necessary since features of the same haplotype might be the causal source of association. It is also critical to exclude the possibility that the association is the result of cryptic relatedness among the cases who share the same variant [174].

7.4. Conclusion

The widespread availability of SNP array data from SNP-based genome-wide association studies can support CNV analysis in the same patients that would potentially provide more insight into the disease susceptibility. However, there is a greater considerable complexity when assessing the role of CNVs in disease that might not be immediately apparent when performing such an analysis. In this study, we have investigated the methodological challenges that emerge in genome-wide association studies of rare CNVs. We have shown that analytical complications can arise from CNV detection and association testing procedures that can pose problems in establishing real rare CNV

associations.

We have illustrated the limited accuracy of CNV detection by measuring the reproducibility between many replicates of one well-characterised sample and how this depends on various quality control criteria. Moreover, we have showed that methods to estimate CNV concordance and frequency in a population can have high effect in association testing procedures. In particular, the CNV burden analysis that has been widely used to test for enrichment of CNVs in cases in comparison to controls can suffer from these issues. Finally, we showed that erroneous CNV associations of specific loci can arise as a result of the *a priori* assumptions employed for testing.

Our knowledge of SNPs and CNVs is constantly changing given the recent and constant development of next-generation sequencing methods. Novel rare SNPs are being reported while CNVs and small insertion/deletion variants are being discovered in high resolution. This can help researchers assess the functional impact of copy number variants as well as evaluate existing results that are based on array data. For instance, the latest studies that have used exome sequencing methods have been supporting a *de novo* paradigm for schizophrenia and autism spectrum disorders [202, 203, 204]. Many previous studies on rare CNVs typed using arrays have been arguing that the paradigm of causality of CNVs could not be viewed simply in the context of whether a CNV is inherited or *de novo*. This might also have been partly due to the limitations of the array technologies of identifying accurately whether a mutation is *de novo* or not. However, these recent findings based on exome sequencing methods indicate that subset of affected individuals might derive their affected status from *de novo* mutations. This is in line with the theory that the pool of causal alleles would have to be replenished by recurrent mutation for these neurodevelopmental diseases to remain in the population, given that affected individuals by these disorders have on average fewer offspring than controls [174].

Box 1 - CNV frequency and region definitions

In order to calculate CNV frequencies or to create CNV regions for testing, one needs to distinguish recurrent events across a population. Multiple rules can be used to accomplish this task, some of which will work better depending on the data quality. The following examples show the resulting calls after frequencies calculations (CNVEs) and regions creation (CNVRs) using sets of CNV events. Three rules are explained, using either any or reciprocal overlap.

Any Overlap ($\mathrm{CNVE_{R\text{-}any}}$ and $\mathrm{CNVR_{R\text{-}any}}$)

The naive way to compute frequencies from CNV events is to create CNV regions, count the number of event inside that region and assign the frequency to each event, according to the total number of samples in the population ($\mathrm{CNVE_{R\text{-}any}}$). This is shown at Figure 7.7, where CNV events and regions are shown in blue and black, respectively. The number of events used to compute the frequency of each underlying CNV event are shown in black. This naive approach allows non-overlapping events to be used to compute the frequency, as every contributing event to the region will have the same frequency. For testing purposes, the boundaries of the CNV region ($\mathrm{CNVR_{R\text{-}any}}$) are shown with dashed lines. These represent the minimum of each CNV's starting positions, and the maximum of each CNV's ending positions.

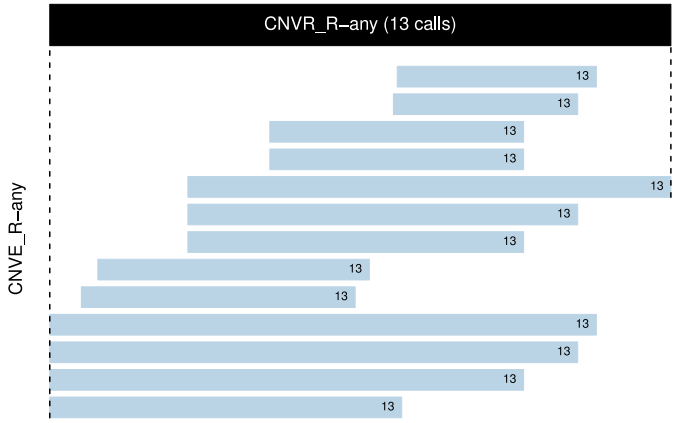


FIGURE 7.7. – Example of $\mathrm{CNVE_{R\text{-}any}}$ and $\mathrm{CNVR_{R\text{-}any}}$. CNV events and CNV region are shown in blue and black, respectively. The number of events used to compute the frequency of each underlying CNV event (with CNV region creation) are shown in black.

Any Overlap ($CNVE_{any}$ and $CNVR_{any}$)

Another naive approach to compute CNV frequencies is to find for each event the number of overlapping events, using any overlap ($CNVE_{any}$). The difference with the first naive approach, is the fact that no CNV regions are created for frequency calculations, as each CNV event is processed one at a time. When creating CNV regions ($CNVR_{any}$) for testing purposes, the frequency of each contributing event might be different, as shown in the following figure, in red. Again, the boundaries of the CNV region are shown with dashed lines, and represent the minimum of each CNV's starting positions, and the maximum of each CNV's ending positions. And advantage of using "any overlap" for the creation of CNV regions is the fact that fragmented calls (within same sample) will be joined together into the same CNV region, even though non-overlapping CNV events from different samples can also be grouped together.

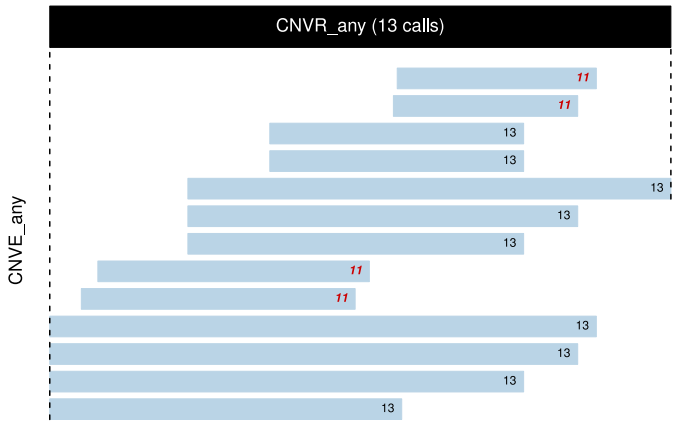


FIGURE 7.8. – Example of $CNVE_{any}$ and $CNVR_{any}$. CNV events and CNV region are shown in blue and black, respectively. The number of events used to compute the frequency of each underlying CNV event (without CNV region creation) are shown in black and red. CNV event sharing the same region might not have the same frequency than the others (red numbers).

Reciprocal Overlap ($CNVE_{50}$, $CNVR_{50}$ and $CNVR_{50.90}$)

The "any overlap" rule rarely works in practise, since long false positive events can alter the region coordinates significantly or might create CNV regions which contain non-overlapping events. Another rule that restrict the amount of overlap between two event might be better for distinguishing recurrent events. In order to calculate the reciprocal overlap, one needs to find the overlapping

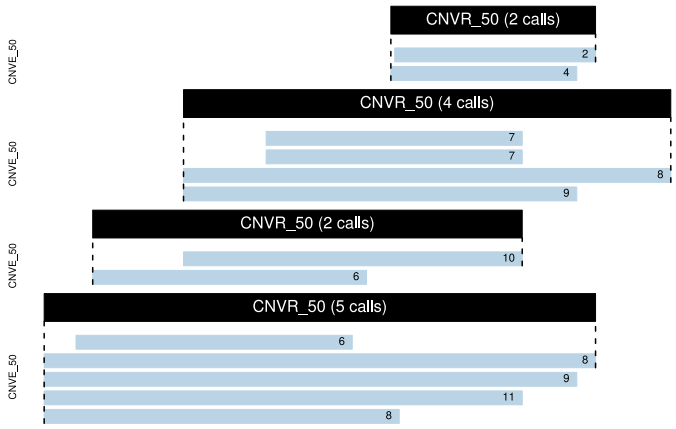


FIGURE 7.10. – Example of $CNVE_{50}$ and $CNVR_{50}$. CNV events and regions are shown in blue and black respectively. Frequencies (black numbers) are computed by creation of regions where each pairwise CNV events overlaps above a threshold (50%).

region, and find the percentage of that region that overlaps each of the two events (see Figure 7.9). Both percentages should be higher than a certain threshold.

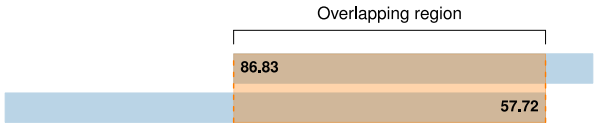


FIGURE 7.9. – Reciprocal overlap. Example of reciprocal overlap between two CNV events (blue). The overlapping region is shown in orange and the numbers represent the overlapping proportion. Both proportions should be above a threshold.

Using the reciprocal overlap with a threshold (such as 50%) ensures that the events ($CNVE_{50}$) have similar length and breakpoints. For testing purposes, CNV regions ($CNVR_{50}$ and $CNVR_{50.90}$) are created by combining CNV events which have all a reciprocal overlap above a certain threshold in between each other. The algorithm used for region creation is based on heuristic rules, hence the final set of regions (and the events composing those regions) are not optimal, and depend on the ordering of CNV events.

There are also multiple ways to determine the starting and ending position of each CNV region. The minimum, and maximum, of all underlying CNV event's starting, and ending position, might be used for the boundaries of the CNV region, as represented with black dashed lines, in the following example ($CNVR_{50}$, using normal boundaries). Also, the region containing at least 90% of the events can be used, as represented with orange dashed lines ($CNVR_{50.90}$, using effective boundaries). In

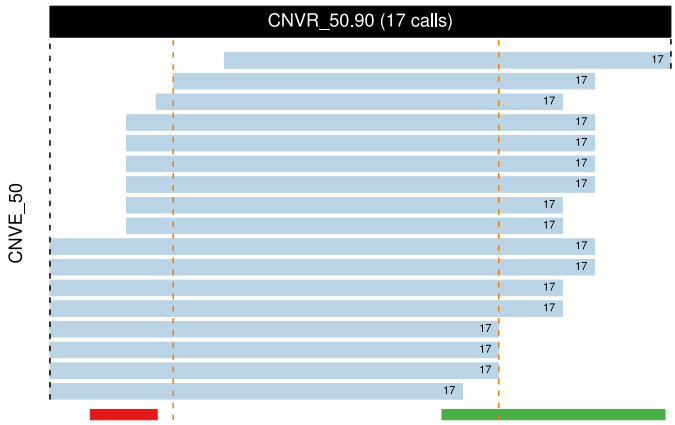


FIGURE 7.11. – Distinction between $CNVR_{50}$ and $CNVR_{50.90}$. Effective breakpoints (**$CNVR_{50.90}$**) correspond to the region where 90% of the CNV events are located.

this example, genes are shown in red and green. If the normal boundaries are used, both genes are considered as been overlapped by the CNV region (containing 17 CNV events). If the effective boundaries are used, only the green gene is considered as been overlapped by the CNV region, as the red one falls outside the area containing at least 90% of the CNV events (16 out of 17 events).

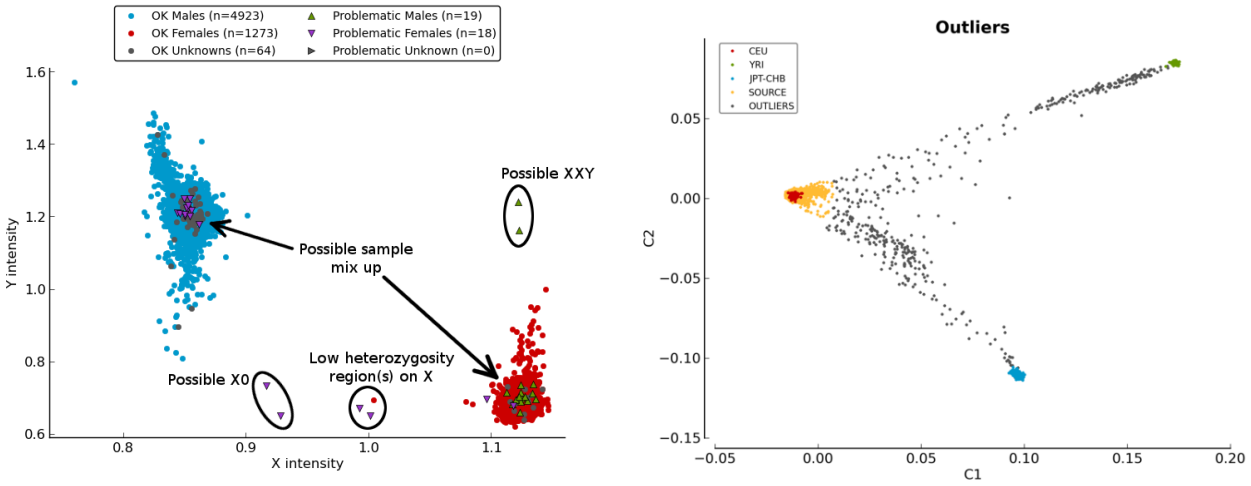
Troisième partie

Conclusions et Perspectives

8. Conclusions

Tel que mentionné au Chapitre 3, l'objectif général de cette thèse vise l'approfondissement des connaissances bio-informatiques dans le domaine des technologies émergentes de la génomique. En particulier, nous avons développé un nouvel outil bio-informatique permettant de filtrer les données génétiques issues de plateformes de génotypage à haut débit dans le but d'obtenir un jeu de données d'analyse de très haute qualité. Ensuite, nous nous sommes intéressés aux nouveaux défis présentés par le génotypage massif de variations nucléotidiques rares ou *de novo* (SNV) rendu possible par l'avancée rapide des technologies de séquençage à haut débit. Plus particulièrement, nous présentons des analyses appropriées lorsque les SNV sont traités à l'aide de méthodes traditionnelles conjointement avec des variations nucléotidiques communes (SNP). Le futur de la génomique reposant sur l'utilisation conjointe d'ensembles bien caractérisés de variations de tout genre et d'outils bio-informatiques robustes et performants, nous avons comparé la performance de différents algorithmes à déterminer le génotype de SNV à l'aide de micropuces d'ADN. Nous nous sommes ensuite intéressés à des variations génomiques souvent méconnues : les variations et les polymorphismes en nombre de copies (CNV et CNP, respectivement). Nous avons ainsi implémenté le premier outil bio-informatique qui, en utilisant l'information provenant des données familiales, permet le partitionnement des CNP afin de les rendre compatibles avec les études de liaison génétique. Finalement, nous avons participé à une étude exploratoire afin d'observer l'impact du choix de la méthode et de la plateforme d'analyse sur les résultats d'études des CNV rares portant sur des populations d'individus non reliés. Tous ces projets ont été réalisés dans le but d'augmenter la compréhension de l'héritabilité des maladies complexes telle la maladie cardiovasculaire, responsable d'une proportion importante des défis en santé auxquels est confrontée la société moderne.

Non seulement est-il important d'étudier le plus de composantes génétiques possible, encore faut-il s'assurer de la qualité des données analysées. Bien que le devis de l'étude soit crucial afin de maximiser la puissance des analyses statistiques subséquentes, les biais technologiques, les biais de recrutement et l'effet de structure de population doivent être diminués au maximum afin de maintenir un taux d'erreur de premier et de second degré minimal. Notre outil nommé *pyGenClean* (Chapitre 4) a été spécialement conçu pour effectuer le nettoyage de données génétiques de manière efficace et reproductible. Bien que notre algorithme utilise un outil souvent employé en génétique (PLINK [128]), ce dernier ne permettait pas de générer automatiquement et facilement un jeu de données de qualité pour les études statistiques. Plusieurs étapes de vérification et de gestion de fichiers (pouvant parfois être très volumineux) sont traditionnellement nécessaires, augmentant ainsi le risque d'erreur humaine à chacune des nombreuses étapes. Notre outil permet l'automatisation et la documentation du nettoyage de données génétiques pour toute étude, quelle que soit sa taille (c.-à-d. le nombre de marqueurs et d'individus). La bonne documentation (voir Annexe III à la page lxv) permet à l'utilisateur de faire un suivi détaillé des filtres et des données de chacun des modules, et régit les étapes nécessitant une intervention décisionnelle de l'utilisateur. De nombreux graphiques aident l'utilisateur à prendre des décisions éclairées et permettent de suivre l'évolution des filtres de nettoyage des données. Par exemple, le module de vérification du genre des individus permet, à l'aide de plusieurs graphiques (p. ex. voir Figure 8.1a) et d'un rapport complet, de trouver des échantillons qui auraient pu être mals placés lors des manipulations en laboratoire. Ce même module permet aussi de détecter des individus ayant des caryotypes spéciaux au niveau des chromosomes sexuels. Un autre module permet d'exclure des individus selon la caractérisation de l'origine génétique afin de diminuer les biais de confusions dus à des sous-structures populationnelles. D'autres exemples d'aides visuelles sont décrits au Chapitre 4 ainsi que dans la documentation de l'outil (Annexe III, page lxv). Finalement, afin d'évaluer la pertinence ainsi que la qualité du nettoyage des données génétiques, deux graphiques de type quantile-quantile (ou *QQ Plot*) ont été générés avant (Figure 8.2a) et après (Figure 8.2b) le nettoyage des données à l'aide de notre outil. On peut clairement voir la présence d'une inflation de la statistique lorsque les données génétiques n'ont pas été nettoyées dans l'exemple donné, en partie dû à la structure populationnelle et à



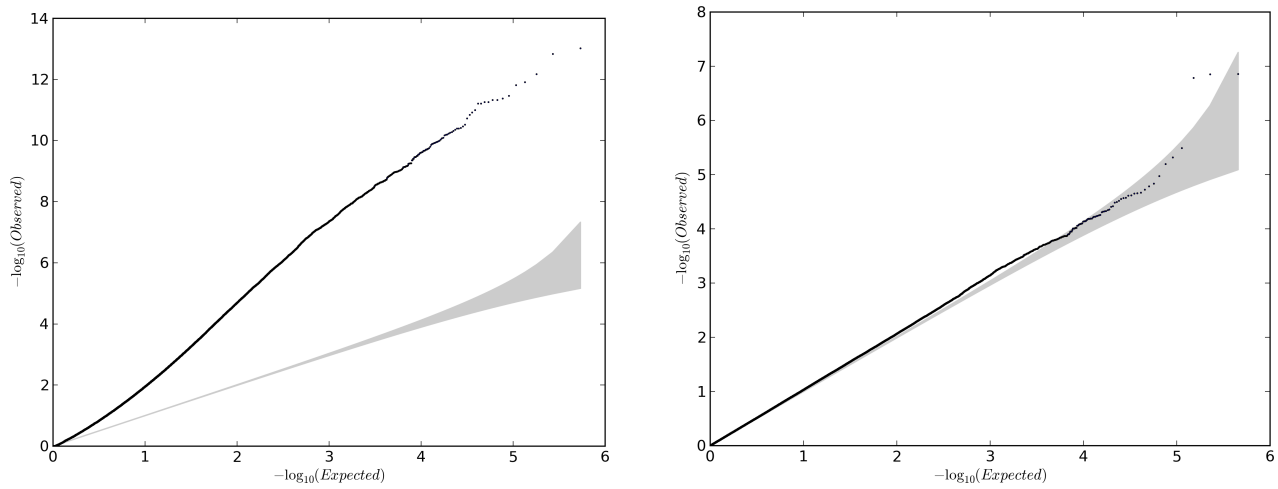
(a) Vérification du genre **(b)** Vérification de l'ethnicité

FIGURE 8.1. – Exemples de résultats visuels créés par l'outil *pyGenClean*. Le premier graphique (8.1a) a été généré par le module de vérification du genre pour 6 297 individus. Les hommes et les femmes se retrouvent à des endroits différents dans le graphique, permettant de distinguer les individus qui auraient pu être mélangés en laboratoire. On peut aussi y apercevoir des individus avec des caryotypes spéciaux sur les chromosomes sexuels (p. ex. XO et XXY). Le deuxième graphique (8.1b) a été généré par le module d'ethnicité. Ce module permet d'exclure des individus dont l'ethnicité est différente de celle attendue. Les individus en gris ont été exclus de l'étude, car ils étaient génétiquement trop différents de la population cible (les Caucasiens, en rouge).

l'usage de marqueurs de moins bonne qualité. Les filtres de qualité utilisés dans le processus de nettoyage permettent de diminuer les biais analytiques et assurent la génération de résultats de meilleure qualité.

Grâce à son implémentation utilisant les normes DRMAA [1], notre outil de nettoyage permet une exécution simple et très rapide des différentes étapes requises. Il a été utilisé dans le cadre d'une étude clinique en pharmacogénomique sur un jeu de données de plus de 6 500 individus et 2 500 000 marqueurs en seulement quatre jours, incluant les vérifications manuelles [126]. Il est aussi très flexible et paramétrable afin de permettre l'usage de filtres de nettoyage sur de plus petits jeux de données comme ceux issus de plateformes de génotypage par Sequenom, comprenant généralement moins d'une centaine de marqueurs génétiques.

1. Distributed Resource Management Application API



(a) Sans nettoyage des données **(b)** Avec nettoyage des données

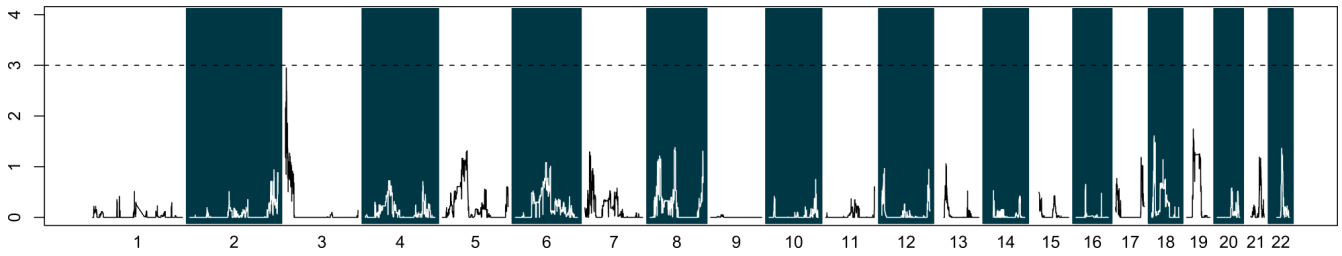
FIGURE 8.2. – Graphique Quantile-Quantile avant et après nettoyage des données génétiques. Ces graphiques ont été générés avant (8.2a) et après (8.2b) le nettoyage des données par l'outil *pyGenClean*. L'inflation des valeurs p, créée par la présence de marqueurs ou d'individus de mauvaise qualité ou encore de sous-structures populationnelles, a été éliminée par le nettoyage des données.

Après les SNP, d'autres marqueurs d'intérêt pour la communauté scientifique sont les variations nucléotidiques rares, présentes dans moins de 1 % de la population à l'étude (SNV). Avec les récentes avancées des plateformes technologiques de séquençage à haut débit, il est maintenant possible de déceler toutes les variations nucléotidiques d'un individu, qu'elles soient rares ou communes. Malheureusement, compte tenu du coût et du temps requis pour chacune des analyses, il demeure peu pratique de séquencer à la fois plusieurs centaines de milliers d'individus. C'est pourquoi nous nous sommes intéressés à la nouvelle micropuce d'ADN de la compagnie Illumina : la HumanExome BeadChip (Chapitre 5). Cette puce permet le génotypage de plusieurs marqueurs rares chez un grand nombre d'individus, et ce, à faible coût. Nous avons donc examiné la performance des algorithmes de génotypage, car leur efficacité à déterminer les génotypes à partir d'une puce contenant une majorité de variations rares était inconnue. Bien que ce type d'analyse ait déjà été fait pour d'autres puces d'ADN contenant une certaine quantité de variations rares (jusqu'à 1 %), il était essentiel de valider les différents algorithmes pour cette nouvelle plateforme contenant une densité de marqueurs rares inégalée. Les algorithmes ont été choisis pour leur capacité prédéfinie à déceler des marqueurs rares. Contrairement à une étude antérieure (Ritchie et collaborateurs [134]), nous avons trouvé que l'algorithme propriétaire d'Illumina (*GenCall* de la suite *GenomeStudio*)

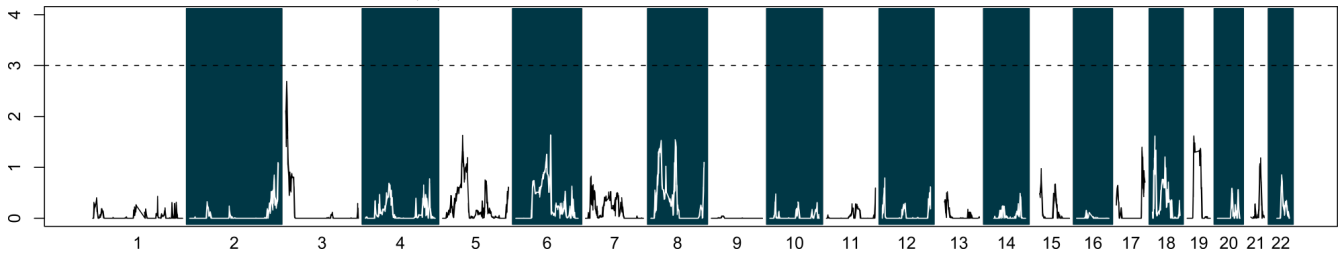
présentait les meilleurs résultats en ce qui a trait à la complétion des données (Section 5.3.2), à la précision (Section 5.3.3) et à l'exactitude des résultats (Section 5.3.4). Il est aussi l'algorithme ayant donné le taux d'erreur le plus faible (Section 5.3.6).

Cette analyse détaillée des algorithmes de génotypage nous a permis de tirer des conclusions importantes. D'abord, qu'un algorithme préalablement favorisé en présence de marqueurs rares se classait dernier lorsque comparé aux autres. En effet, nous avons dû modifier les paramètres du code source de l'algorithme de *GenoSNP* afin d'en augmenter sa performance. Nous avons aussi pu trancher relativement à la nécessité d'utiliser une optimisation manuelle avec l'algorithme propriétaire d'Illumina, une approche traditionnellement prisée, mais souvent négligée puisqu'exigeante. Notre étude a ainsi permis de conclure avec confiance que l'optimisation manuelle a rendu possible un gain important de marqueurs analysables sans compromettre la précision et l'exactitude des résultats.

Lorsque l'intérêt de la communauté scientifique s'est tourné vers les CNV et les CNP, les micropuces d'ADN furent l'approche technologique de prédilection pour en faire la détection. Cette technologie ne requérant aucuns frais de développement supplémentaires, puisque les données nécessaires étaient déjà produites pour l'étude très répandue des SNP (c.-à-d. les intensités de chacune des sondes). Plusieurs algorithmes ont donc été créés afin de permettre la détection et l'analyse de ces nouveaux marqueurs. Bien que plusieurs outils aient été conçus dans le passé afin de réaliser des études d'associations avec les CNP, aucune méthode n'existait pour la réalisation d'études de liaison génétique sur données familiales. L'un des projets présenté dans cette thèse a donc été la création du premier algorithme permettant le partitionnement des CNP sur chacune des paires de chromosomes afin de suivre la transmission de ces variations structurales d'un parent à sa descendance (algorithme *CNGen,* Chapitre 6). L'algorithme, publié dans le journal *BMC Bioinformatics*, a été validé à l'aide de simulations et sa robustesse contre les erreurs de génotypage ou la présence de variations *de novo* chez les individus a été prouvée. Un autre avantage de *CNGen* est qu'il permet la création efficace d'haplotypes à partir des CNP partitionnés et ce, à l'aide d'outils déjà existants. Une limite de notre



(a) Étude de liaison génétique avec SNP



(b) Étude de liaison génétique avec CNP partitionnés

FIGURE 8.3. – Résultats d'études de liaison génétique à l'aide des SNP et des CNP. Les deux études de liaison génétique ont produit des résultats concordants entre l'analyse avec SNP (8.3a) et CNP partitionnés (8.3b). La région 3p26 a été soulignée dans les deux cas.

méthode est la supposition que l'expansion en nombre de copies est faite à partir du même allèle ancestral. Bien que les méthodes de création des CNP suggèrent que c'est effectivement le cas pour la majorité des CNP, il est possible que des CNP soient créés lors d'évènement de recombinaison, ouvrant ainsi la possibilité à une expansion d'allèle de type différent. Ces CNP hétérogènes seraient décelés par les filtres de validation mendélienne nécessaires à l'utilisation de l'algorithme *CNGen*, et seraient de ce fait retirés des analyses.

L'algorithme *CNGen* a aussi été utilisé avec succès sur des données réelles provenant de familles québécoises dont certains individus étaient atteints d'une maladie cardiaque congénitale affectant le côté gauche du cœur (*left ventricular outflow tract obstruction* ou LVOTO). Afin de mieux comprendre l'étiologie de cette maladie, deux études de liaison génétique ont été réalisées : l'une utilisant des marqueurs traditionnels (c.-à-d. des SNP) et l'autre, des CNP nouvellement partitionnés par notre algorithme. Les résultats des deux études se sont montrés très similaires, soulignant la même région génomique sur le chromosome 3 avec un taux de signification suggestif tout à fait comparable (Figure 8.3, manuscrit en préparation, troisième auteur). Ces résultats nous ont amenés

à tirer une conclusion importante relativement à la transmission des CNP : que les CNP seront en déséquilibre de liaison avec les SNP (voir la Section 1.3.3 pour plus d'information sur le déséquilibre de liaison entre les SNP et les CNP communs) et que les analyses de liaison génétique par SNP peuvent être garantes des variations CNP.

Toutefois, nous avons pu démontrer qu'une approche combinant à la fois l'information sur les CNV issus de transmission familiale à celle sur les CNV issus de cohortes observationnelles peut apporter un gain analytique important. En effet, cette étude a permis de générer une liste de CNV rares de bonne qualité, menant à l'identification de régions génomiques liées aux maladies congénitales du côté gauche du coeur et, par le fait même, à la détection de 25 gènes candidats dont *SMC1A*, *EVC2* (Figure 8.4), *MFAP4* et *CTHRC1*, tous chevauchant des régions connues de syndromes [14].

Grâce à l'outil bio-informatique *CNGen*, nous avons rendu possible l'utilisation des variations en nombre de copies dans le cadre d'études de liaison génétique. Nous avons aussi pu déterminer, avec l'aide des données familiales, quels sont les paramètres nécessaires afin d'obtenir un ensemble de variations de haute qualité (c.-à-d. en utilisant les informations d'héritabilité des variations structurales). Un défi important restait néanmoins : déterminer l'impact de ces paramètres dans l'étude d'individus non reliés (études composées de cas et de témoins). C'est avec la collaboration du *Wellcome Trust Center for Human Genetics* de l'*University of Oxford* que nous avons entrepris une étude exploratrice portant sur les variations structurales rares (CNV) afin d'observer les différences entre les ensembles de variations résultant de l'utilisation de plateformes et de méthodes d'analyses différentes (Chapitre 7). À partir de données provenant de deux plateformes distinctes et de plusieurs milliers d'individus témoins issus d'une population référence, nous avons démontré hors de tout doute que le choix de la plateforme de génotypage peut avoir un grand impact sur le taux de faux positifs et de faux négatifs des CNV. De plus, les pratiques courantes de filtration des données (p. ex. la taille ou le seuil de chevauchement) ont un impact important sur le taux de faux négatifs à grande échelle. Nous avons aussi déterminé que le choix de la méthode permettant de déterminer la fréquence de chacune des variations trouvées (c.-à-d. la méthode de chevauchement)

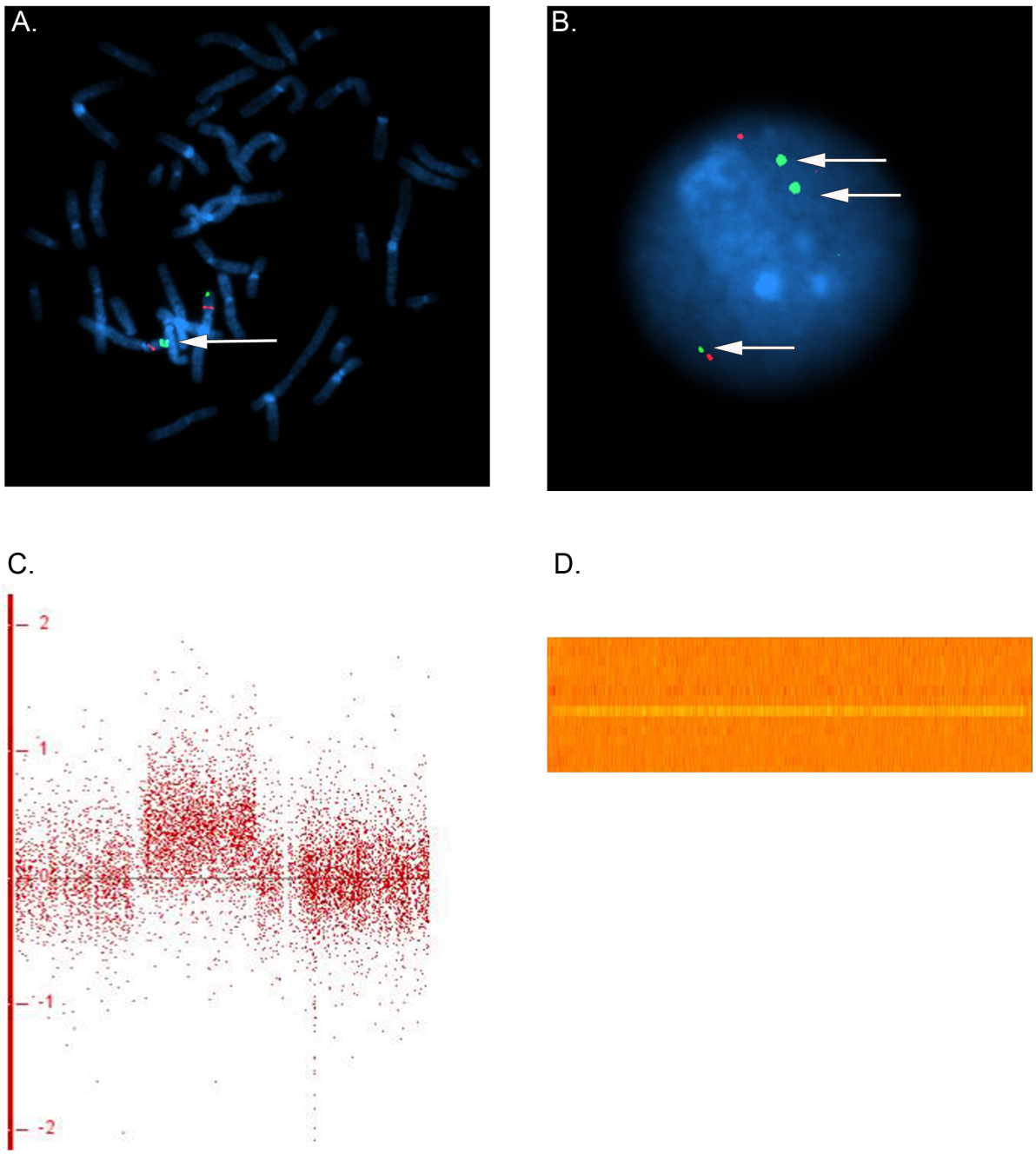


FIGURE 8.4. – Duplication sur le chromosome 4 englobant 34 gènes, dont *EVC2*. CNV représenté par un gain de 3 817 kb situé sur le chromosome 4 chez une mère. Deux hybridations *in situ* en fluorescence ont confirmé la présence de la duplication dans les chromosomes en métaphase de la mère (a, points verts) ainsi que dans le noyau d'une cellule extraite (b, trois points verts). De plus, deux évidences bio-informatiques (augmentation du log R ratio [c], ainsi qu'une augmentation de l'intensité des sondes dans la région dupliquée, visible dans un heatmap [d]), confirment la présence de ce gain chez la mère. Figure adaptée de Hitz et collaborateurs [14].

a un impact majeur sur les résultats des analyses subséquentes, car elle modifie considérablement la liste de variations finale. Ceci est dû au fait que les méthodes de détection des CNV utilisant les micropuces d'ADN ne sont pas très précises en ce qui a trait aux extrémités et qu'il est difficile de déterminer avec certitude les régions qui varient en nombre de copies. Ce projet de recherche est encore d'actualité. Effectivement, bien que les nouvelles technologies de séquençage à haut débit promettent d'améliorer la précision des extrémités des CNV, une étude en cours dans notre laboratoire montre que plusieurs facteurs peuvent affecter les résultats obtenus et qu'une grande quantité de bruit peut être présente dans les résultats.

Ces observations suscitent un questionnement important, compte tenu du nombre important de publications scientifiques faisant état d'associations avec des CNV rares. On note qu'une grande majorité de ces publications ne font pas état des paramètres de définition de CNV utilisés ni de la validité interne des résultats propres à ces paramètres. En collaboration avec le *Wellcome Trust Centre for Human Genetics*, nous avons revisité les analyses présentées dans l'étude de la schizophrénie de Vacic et collaborateurs [16]. Les auteurs de l'article y rapportent des évènements de duplications rares près du gène *VIPR2*, qui sont plus communes chez des individus d'origine diverse (hispanique, caucasienne et africaine-américaine) atteints de schizophrénie que chez des individus témoins avec une valeur p de $5,7 \times 10^{-7}$. Cette valeur significative a été obtenue pour des CNV de duplication présents en excès chez des cas (0,35 % ; 29 sur 8 290) versus des témoins (0,027 % ; 2 sur 7 431). Nous avons reconstruit les CNV de 7 331 individus issus d'une population référence d'origine anglaise. Nous y avons trouvé 0,11 % de duplications au gène *VIPR2* (Tableau VIII.I), soit quatre fois plus que rapporté dans l'article de Vacic et collaborateurs. Cette différence peut être expliquée par des différences de paramétrisation telles l'approche de normalisation des données, le choix de l'algorithme de génotypage, les seuils de qualités utilisés, les définitions de variation rare utilisées, le seuil de longueur utilisé, ou encore la définition d'un chevauchement utilisée. Par exemple, lorsque nous considérons les duplications dont la longueur est supérieure à 50 kb (versus 100 kb), nous observerons alors trois duplications de plus dans notre population de référence. De plus, selon nos analyses, nous identifions trois délétions dans la région du gène, alors que les auteurs

Table VIII.I. – Duplications chevauchant le gène *VIPR2*. Comparaison de l'évidence d'association de duplications dans la région du gène *VIPR2* et la schizophrénie (Vacic et collaborateurs [16]). Aussi, l'information sur les duplications trouvées chez 7 331 individus témoins dans notre étude (cellules grises). Un test de Fisher exact a été utilisé afin de calculer les valeurs p entre les cas et les deux groupes de témoins. Toutes les duplications trouvées ont été validées par une autre plateforme et un autre algorithme. La région combinée comprend à la fois la région proximale (de 158 448 321 à 158 605 936) et distale (de 158 731 401 à 158 810 016) du gène *VIPR2*, sur le chromosome 7.

Nom de la région	**Cas (%)** ($n = 8\,290$)		**Témoins (%)** ($n = 7\,431$)		**Valeur P reportée**	**Témoins (%)** ($n = 7\,331$)		**Notre valeur P**
Combinée	29	(0,35 %)	2	(0,027 %)	$5,7 \times 10^{-7}$	8	(0,11 %)	0,001317
Proximale	20	(0,24 %)	1	(0,013 %)	0,0007	5	(0,068 %)	0,005036
Distale	14	(0,17 %)	0	(0 %)	4×10^{-5}	3	(0,041 %)	0,03181

n'en ont trouvé aucune. Cette réanalyse des données de Vacic et collaborateurs nous force à conclure que les paramètres de CNV utilisés peuvent avoir un impact majeur sur les résultats d'associations génétiques. Une petite différence de paramétrisation entre les groupes comparés mènera alors à de fausses associations. Pour les CNV rares, un taux de 0,027 % représente alors deux duplications dans la région du gène et une simple augmentation de quatre variations peut faire passer la valeur p de $5,7 \times 10^{-7}$ à $1,317 \times 10^{-3}$ (Tableau VIII.I). Nous concluons ainsi qu'il est essentiel de documenter et de valider l'impact des paramètres définissant les CNV afin de garantir la validité des résultats rapportés.

Le problème de l'héritabilité manquante des maladies complexes en est un de taille, qui peut être approché par plusieurs angles. Cette thèse aura pu contribuer en partie à l'amélioration de la qualité des données, entraînant ainsi une puissance de détection accrue et une diminution de faux positifs des études génétiques. Par le développement d'outils puissants, nous avons rendu possible l'analyse des CNV et des CNP dans le cadre d'étude de liaison génétique. Nous avons aussi permis de mieux comprendre les effets du choix d'un devis d'étude lors de l'analyse des CNV dans des populations d'individus non reliés. Finalement, que ce soit par le génotypage adéquat de variations rares ou par le nettoyage précis du jeu de données génétiques, nous avons permis la création d'ensembles de données incluant des variations nucléotidiques rares et communes nécessaires à l'obtention d'une puissance adéquate pour les tests statistiques impliqués dans l'étude des maladies complexes. La

bio-informatique est un domaine en plein essor qui est voué à un rôle important dans un monde où la quantité de données ainsi que la complexité des analyses requises ne cesseront d'augmenter.

9. Perspectives

Les travaux présentés dans cette thèse ont permis d'approfondir nos connaissances sur les nouvelles technologies applicables à la génomique, tout en offrant à la communauté scientifique des outils bio-informatiques de qualité.

Bien que les micropuces d'ADN soient un moyen rapide, efficace et peu coûteux permettant le génotypage de millions de variations nucléotidiques communes (et maintenant rares), l'ensemble des marqueurs ciblés ne représente qu'une petite partie de toutes les mutations pathogéniques possibles chez l'homme. Afin de pouvoir examiner les mutations plus rares, voire personnelles à un génome en particulier, la génomique doit se tourner vers le séquençage à haut débit. En dépit du fait que ces technologies soient grandement utilisées depuis les quelques dernières années, plusieurs régions du génome restent sous-étudiées à cause de leurs caractéristiques. Ces régions chevauchent souvent des gènes d'intérêts, tels que les gènes à forte homologie, dont ceux impliqués dans le métabolisme des médicaments. Ainsi, le séquençage à haut débit est un sujet d'intérêt bien particulier qui nécessitera des développements bio-informatiques adaptés et d'importance afin d'atteindre une médecine personnalisée efficace et sécuritaire.

Encore beaucoup de chemin reste à faire afin d'augmenter les connaissances sur l'étiologie génétique des maladies. Bien que les variations en nombre de copies (rares ou communes) et des variations nucléotidiques rares aient permis de franchir un pas de plus, d'autres types de marqueurs génomiques gagnent encore à être explorés. L'épigénétique en est un exemple, où des différences dans l'expression de certains gènes sont engendrées non pas par des variations dans la séquence d'ADN du

génome, mais bien par l'environnement et l'histoire individuelle de chacun des individus.

Finalement, le défi ultime à relever pour accéder à une compréhension approfondie de l'héritabilité réside dans l'intégration des données en un unique outil statistique puissant et efficace. Depuis plusieurs années, la communauté scientifique travaille à combiner plusieurs technologies ou concepts, comme les interactions entre la génétique et l'environnement ou encore, l'étude des variations nucléotidiques rares et communes. Le grand défi consiste à assembler toutes ces informations (environnement, génétique, protéomique, épigénétique, expression, etc.) et à mieux comprendre le cadre régissant l'interdépendance de ces concepts. Par la création de jeux de données de haute qualité et par l'accroissement des connaissances sur chacun de ces concepts, la communauté scientifique se verra mieux outillée pour l'avenir de la génomique et les éventuels défis qu'elle nous réserve.

Bibliographie

[1] Beckmann JS, Estivill X, Antonarakis SE: **Copy number variants and genetic traits: closer to the resolution of phenotypic to genotypic variability**. *Nature Reviews Genetics* 2007, **8**(8):639–646. [DOI:10.1038/nrg2149].

[2] Syvänen AC: **Accessing genetic variation: genotyping single nucleotide polymorphisms**. *Nature Reviews Genetics* 2001, **2**(12):930–942. [DOI:10.1038/35103535].

[3] The 1000 Genomes Project Consortium: **A map of human genome variation from population-scale sequencing**. *Nature* 2010, **467**(7319):1061–1073. [DOI:10.1038/nature09534].

[4] Lupski JR: **Genomic disorders: structural features of the genome can lead to DNA rearrangements and human disease traits**. *Trends in genetics* 1998, **14**(10):417–422. [DOI:10.1016/S0168-9525(98)01555-8].

[5] Liu P, Carvalho C, Hastings P, Lupski JR: **Mechanisms for recurrent and complex human genomic rearrangements**. *Current Opinion in Genetics & Development* 2012, **22**(3):211–220. [DOI:10.1016/j.gde.2012.02.012].

[6] Lee JA, Carvalho C, Lupski JR: **A DNA replication mechanism for generating nonrecurrent rearrangements associated with genomic disorders**. *Cell* 2007, **131**(7):1235–1247. [DOI:10.1016/j.cell.2007.11.037].

[7] Hastings P, Ira G, Lupski JR: **A microhomology-mediated break-induced replication model for the origin of human copy number variation**. *PLoS Genetics* 2009, **5**:e1000327. [DOI:10.1371/journal.pgen.1000327].

[8] Korbel JO, Urban AE, Affourtit JP, Godwin B, Grubert F, Simons JF, Kim PM, Palejev D, Carriero NJ, Du L, et al.: **Paired-end mapping reveals extensive structural variation in the human genome**. *Science* 2007, **318**(5849):420–426. [DOI:10.1126/science.1149504].

[9] MacDonald JR, Ziman R, Yuen RK, Feuk L, Scherer SW: **The Database of Genomic Variants: a curated collection of structural variation in the human genome**. *Nucleic Acids Research* 2014, **42**:D986–D992. [DOI:10.1093/nar/gkt958].

[10] Teare MD, Barrett JH: **Genetic linkage studies**. *The Lancet* 2005, **366**(9490):1036–1044. [DOI:10.1016/S0140-6736(05)67382-5].

[11] Hirschhorn JN, Lohmueller K, Byrne E, Hirschhorn K: **A comprehensive review of genetic association studies**. *Genetics in Medicine* 2002, **4**(2):45–61. [DOI:10.1097/00125817-200203000-00002].

[12] Liu N, Zhang D, Zhao H: **Genotyping error detection in samples of unrelated individuals without replicate genotyping**. *Human Heredity* 2009, **67**(3):154–162. [DOI:10.1159/000181153].

[13] Mäkinen VP, Parkkonen M, Wessman M, Groop PH, Kanninen T, Kaski K: **High-throughput pedigree drawing**. *European Journal of Human Genetics* 2005, **13**(8):987–989. [DOI:10.1038/sj.ejhg.5201430].

[14] Hitz MP, Lemieux-Perreault LP, Marshall C, Feroz-Zada Y, Davies R, Yang SW, Lionel AC, D'Amours G, Lemyre E, Cullum R, et al.: **Rare copy number variants contribute to congenital left-sided heart disease**. *PLoS Genetics* 2012, **8**(9):e1002903. [DOI:10.1371/journal.pgen.1002903].

[15] Liu P, Lacaria M, Zhang F, Withers M, Hastings P, Lupski JR: **Frequency of nonallelic homologous recombination is correlated with length of homology: evidence that ectopic synapsis precedes ectopic crossing-over**. *The American Journal of Human Genetics* 2011, **89**(4):580–588. [DOI:10.1016/j.ajhg.2011.09.009].

[16] Vacic V, McCarthy S, Malhotra D, Murray F, Chou HH, Peoples A, Makarov V, Yoon S, Bhandari A, Corominas R, et al.: **Duplications of the neuropeptide receptor gene VIPR2 confer significant risk for schizophrenia**. *Nature* 2011, **471**(7339):499–503. [DOI:10.1038/nature09884].

[17] Lander ES, Linton LM, Birren B, Nusbaum C, Zody MC, Baldwin J, Devon K, Dewar K, Doyle M, FitzHugh W, et al.: **Initial sequencing and analysis of the human genome**. *Nature* 2001, **409**(6822):860–921. [DOI:10.1038/35057062].

[18] Li WH, Sadler LA: **Low nucleotide diversity in man**. *Genetics* 1991, **129**(2):513–523.

[19] The 1000 Genomes Project Consortium, Abecasis GR, Auton A, Brooks LD, DePristo MA, Durbin RM, Handsaker RE, Kang HM, Marth GT, McVean GA, et al.: **An integrated map of genetic variation from 1,092 human genomes**. *Nature* 2012, **491**(7422):56–65. [DOI:10.1038/nature11632].

[20] Conrad DF, Keebler JE, DePristo MA, Lindsay SJ, Zhang Y, Casals F, Idaghdour Y, Hartl CL, Torroja C, Garimella KV, et al.: **Variation in genome-wide mutation rates within and between human families**. *Nature Genetics* 2011, **43**(7):712–714. [DOI:10.1038/ng.862].

[21] Kan YW, Dozy AM: **Polymorphism of DNA sequence adjacent to human beta-globin structural gene: relationship to sickle mutation**. *Proceedings of the National Academy of Sciences* 1978, **75**(11):5631–5635.

[22] Wyman AR, White R: **A highly polymorphic locus in human DNA**. *Proceedings of the National Academy of Sciences* 1980, **77**(11):6754–6758.

[23] Jeffreys AJ, Wilson V, Thein SL, et al.: **Hypervariable'minisatellite' regions in human DNA**. *Nature* 1985, **314**(6006):67–73.

[24] Litt M, Luty JA: **A hypervariable microsatellite revealed by in vitro amplification of a dinucleotide repeat within the cardiac muscle actin gene**. *American Journal of Human Genetics* 1989, **44**(3):397.

[25] Kan Y, AM D, Varmus H, Taylor J, Holland J, Lie-Injo L, Ganesan J, Todd D: **Deletion of α-globin genes in haemoglobin-H disease demonstrates multiple α-globin structural loci**. *Nature* 1975, **255**(5505):255–256.

[26] Goossens M, Dozy AM, Embury SH, Zachariades Z, Hadjiminas MG, Stamatoyannopoulos G, Kan YW: **Triplicated alpha-globin loci in humans**. *Proceedings of the National Academy of Sciences* 1980, **77**:518–521.

[27] Iafrate AJ, Feuk L, Rivera MN, Listewnik ML, Donahoe PK, Qi Y, Scherer SW, Lee C: **Detection of large-scale variation in the human genome**. *Nature Genetics* 2004, **36**(9):949–951. [DOI:10.1038/ng1416].

[28] Sebat J, Lakshmi B, Troge J, Alexander J, Young J, Lundin P, Månér S, Massa H, Walker M, Chi M, et al.: **Large-scale copy number polymorphism in the human genome**. *Science* 2004, **305**(5683):525–528. [DOI:10.1126/science.1098918].

[29] Kruglyak L, Nickerson DA: **Variation is the spice of life**. *Nature Genetics* 2001, **27**(3):234–236. [DOI:10.1038/85776].

[30] Reich DE, Gabriel SB, Altshuler D: **Quality and completeness of SNP databases**. *Nature genetics* 2003, **33**(4):457–458. [DOI:10.1038/ng1133].

[31] The International HapMap Consortium: **The international HapMap project**. *Nature* 2003, **426**(6968):789–796. [DOI:10.1038/nature02168].

[32] Gunderson KL, Steemers FJ, Ren H, Ng P, Zhou L, Tsan C, Chang W, Bullis D, Musmacker J, King C, et al.: **Whole-genome genotyping**. *Methods in Enzymology* 2006, **410**:359. [DOI:10.1016/S0076-6879(06)10017-8].

[33] Gunderson KL, Steemers FJ, Lee G, Mendoza LG, Chee MS: **A genome-wide scalable SNP genotyping assay using microarray technology**. *Nature Genetics* 2005, **37**(5):549–554. [DOI:10.1038/ng1547].

[34] Steemers FJ, Chang W, Lee G, Barker DL, Shen R, Gunderson KL, et al.: **Whole-genome genotyping with the single-base extension assay**. *Nature Methods* 2006, **3**:31–33. [DOI:10.1038/nmeth842].

[35] Rusk N: **Torrents of sequence**. *Nature Methods* 2011, **8**:44–44. [DOI:10.1038/nmeth.f.330].

[36] Metzker ML: **Sequencing technologies—the next generation**. *Nature Reviews Genetics* 2010, **11**:31–46. [DOI:10.1038/nrg2626].

[37] Fonseca NA, Rung J, Brazma A, Marioni JC: **Tools for mapping high-throughput sequencing data**. *Bioinformatics* 2012, **28**(24):3169–3177. [DOI:10.1093/bioinformatics/bts605].

[38] Mardis ER: **Next-Generation Sequencing Platforms**. *Annual Review of Analytical Chemistry* 2013, **6**:287–303. [DOI:10.1146/annurev-anchem-062012-092628].

[39] Browning SR, Browning BL: **Haplotype phasing: existing methods and new developments**. *Nature Reviews Genetics* 2011, **12**(10):703–714. [DOI:10.1038/nrg3054].

[40] Ku CS, Roukos DH: **From next-generation sequencing to nanopore sequencing technology: paving the way to personalized genomic medicine**. *Expert Review of Medical Devices* 2013, **10**:1–6. [DOI:10.1586/erd.12.63].

[41] Shapiro E, Biezuner T, Linnarsson S: **Single-cell sequencing-based technologies will revolutionize whole-organism science**. *Nature Reviews Genetics* 2013, **14**(9):618–630. [DOI:10.1038/nrg3542].

[42] The International HapMap Consortium: **A haplotype map of the human genome**. *Nature* 2005, **437**(7063):1299–1320. [DOI:10.1038/nature04226].

[43] The International HapMap Consortium: **A second generation human haplotype map of over 3.1 million SNPs**. *Nature* 2007, **449**(7164):851–861. [DOI:10.1038/nature06258].

[44] The International HapMap 3 Consortium: **Integrating common and rare genetic variation in diverse human populations**. *Nature* 2010, **467**(7311):52–58. [DOI:10.1038/nature09298].

[45] Feuk L, Carson AR, Scherer SW: **Structural variation in the human genome**. *Nature Reviews Genetics* 2006, **7**(2):85–97. [DOI:10.1038/nrg1767].

[46] Redon R, Ishikawa S, Fitch KR, Feuk L, Perry GH, Andrews TD, Fiegler H, Shapero MH, Carson AR, Chen W, et al.: **Global variation in copy number in the human genome**. *Nature* 2006, **444**(7118):444–454. [DOI:10.1038/nature05329].

[47] Wong KK, deLeeuw RJ, Dosanjh NS, Kimm LR, Cheng Z, Horsman DE, MacAulay C, Ng RT, Brown CJ, Eichler EE, et al.: **A comprehensive analysis of common copy-number variations in the human genome**. *The American Journal of Human Genetics* 2007, **80**:91–104. [DOI:10.1086/510560].

[48] Conrad DF, Pinto D, Redon R, Feuk L, Gokcumen O, Zhang Y, Aerts J, Andrews TD, Barnes C, Campbell P, et al.: **Origins and functional impact of copy number variation in the human genome**. *Nature* 2010, **464**(7289):704–712. [DOI:10.1038/nature08516].

[49] Hastings P, Lupski JR, Rosenberg SM, Ira G: **Mechanisms of change in gene copy number**. *Nature Reviews Genetics* 2009, **10**(8):551–564. [DOI:10.1038/nrg2593].

[50] Zhang F, Gu W, Hurles ME, Lupski JR: **Copy number variation in human health, disease, and evolution**. *Annual Review of Genomics and Human Genetics* 2009, **10**:451–481. [DOI:10.1146/annurev.genom.9.081307.164217].

[51] Lupski JR, Stankiewicz P: **Genomic disorders: molecular mechanisms for rearrangements and conveyed phenotypes**. *PLoS Genetics* 2005, **1**(6):e49. [DOI:10.1371/journal.pgen.0010049].

[52] Inoue K, Lupski JR: **Molecular mechanisms for genomic disorders**. *Annual Review of Genomics and Human Genetics* 2002, **3**:199–242. [DOI:10.1146/annurev.genom.3.032802.120023].

[53] Kleinjan DA, van Heyningen V: **Long-range control of gene expression: emerging mechanisms and disruption in disease**. *The American Journal of Human Genetics* 2005, **76**:8–32. [DOI:10.1086/426833].

[54] Närvä E, Autio R, Rahkonen N, Kong L, Harrison N, Kitsberg D, Borghese L, Itskovitz-Eldor J, Rasool O, Dvorak P, et al.: **High-resolution DNA analysis of human embryonic stem cell lines reveals culture-induced copy number changes and loss of heterozygosity**. *Nature Biotechnology* 2010, **28**(4):371–377. [DOI:10.1038/nbt.1615].

[55] Gu W, Zhang F, Lupski JR: **Mechanisms for human genomic rearrangements**. *Pathogenetics* 2008, **1**:4. [DOI:10.1186/1755-8417-1-4].

[56] Stankiewicz P, Lupski JR, et al.: **Genome architecture, rearrangements and genomic disorders**. *Trends in Genetics* 2002, **18**(2):74–82. [DOI:10.1016/S0168-9525(02)02592-1].

[57] Turner DJ, Miretti M, Rajan D, Fiegler H, Carter NP, Blayney ML, Beck S, Hurles ME: **Germline rates of de novo meiotic deletions and duplications causing several genomic disorders**. *Nature Genetics* 2008, **40**:90–95. [DOI:10.1038/ng.2007.40].

[58] Lieber MR, Ma Y, Pannicke U, Schwarz K: **Mechanism and regulation of human non-homologous DNA end-joining**. *Nature Reviews Molecular Cell Biology* 2003, **4**(9):712–720. [DOI:10.1038/nrm1202].

[59] Zhang F, Carvalho C, Lupski JR, et al.: **Complex human chromosomal and genomic rearrangements**. *Trends in Genetics* 2009, **25**(7):298–307. [DOI:10.1016/j.tig.2009.05.005].

[60] Zhang F, Khajavi M, Connolly AM, Towne CF, Batish SD, Lupski JR: **The DNA replication FoSTeS/MMBIR mechanism can generate genomic, genic and exonic complex rearrangements in humans**. *Nature Genetics* 2009, **41**(7):849–853. [DOI:10.1038/ng.399].

[61] Kallioniemi A, Kallioniemi OP, Sudar Da, Rutovitz D, Gray JW, Waldman F, Pinkel D: **Comparative genomic hybridization for molecular cytogenetic analysis of solid tumors**. *Science* 1992, **258**(5083):818–821. [DOI:10.1126/science.1359641].

[62] Bentz M, Plesch A, Stilgenbauer S, Döhner H, Lichter P: **Minimal sizes of deletions detected by comparative genomic hybridization**. *Genes, Chromosomes and Cancer* 1998, **21**(2):172–175. [DOI:10.1002/(SICI)1098-2264(199802)21:2<172::AID-GCC14>3.0.CO;2-T].

[63] Piper J, Rutovitz D, Sudar D, Kallioniemi A, Kallioniemi OP, Waldman FM, Gray JW, Pinkel D: **Computer image analysis of comparative genomic hybridization**. *Cytometry* 1995, **19**:10–26. [DOI:10.1002/cyto.990190104].

[64] Albertson DG, Pinkel D: **Genomic microarrays in human genetic disease and cancer**. *Human Molecular Genetics* 2003, **12**(suppl 2):R145–R152. [DOI:10.1093/hmg/ddg261].

[65] Korn JM, Kuruvilla FG, McCarroll SA, Wysoker A, Nemesh J, Cawley S, Hubbell E, Veitch J, Collins PJ, Darvishi K, et al.: **Integrated genotype calling and association analysis of SNPs, common copy number polymorphisms and rare CNVs**. *Nature Genetics* 2008, **40**(10):1253–1260. [DOI:10.1038/ng.237].

[66] Speicher MR, Carter NP: **The new cytogenetics: blurring the boundaries with molecular biology**. *Nature Reviews Genetics* 2005, **6**(10):782–792. [DOI:10.1038/nrg1692].

[67] Fridlyand J, Snijders AM, Pinkel D, Albertson DG, Jain AN: **Hidden Markov models approach to the analysis of array CGH data**. *Journal of Multivariate Analysis* 2004, **90**:132–153. [DOI:10.1016/j.jmva.2004.02.008].

[68] Wang P, Kim Y, Pollack J, Narasimhan B, Tibshirani R: **A method for calling gains and losses in array CGH data**. *Biostatistics* 2005, **6**:45–58. [DOI:10.1093/biostatistics/kxh017].

[69] Olshen AB, Venkatraman E, Lucito R, Wigler M: **Circular binary segmentation for the analysis of array-based DNA copy number data**. *Biostatistics* 2004, **5**(4):557–572. [DOI:10.1093/biostatistics/kxh008].

[70] Venkatraman E, Olshen AB: **A faster circular binary segmentation algorithm for the analysis of array CGH data**. *Bioinformatics* 2007, **23**(6):657–663. [DOI:10.1093/bioinformatics/btl646].

[71] Hupé P, Stransky N, Thiery JP, Radvanyi F, Barillot E: **Analysis of array CGH data: from signal ratio to gain and loss of DNA regions**. *Bioinformatics* 2004, **20**(18):3413–3422. [DOI:10.1093/bioinformatics/bth418].

[72] Fiegler H, Redon R, Andrews D, Scott C, Andrews R, Carder C, Clark R, Dovey O, Ellis P, Feuk L, et al.: **Accurate and reliable high-throughput detection of copy number variation in the human genome**. *Genome Research* 2006, **16**(12):1566–1574. [DOI:10.1101/gr.5630906].

[73] Lucito R, Healy J, Alexander J, Reiner A, Esposito D, Chi M, Rodgers L, Brady A, Sebat J, Troge J, et al.: **Representational oligonucleotide microarray analysis: a high-resolution method to detect genome copy number variation**. *Genome Research* 2003, **13**(10):2291–2305. [DOI:10.1101/gr.1349003].

[74] Wang K, Li M, Hadley D, Liu R, Glessner J, Grant SF, Hakonarson H, Bucan M: **PennCNV: an integrated hidden Markov model designed for high-resolution copy number variation detection in whole-genome SNP genotyping data**. *Genome Research* 2007, **17**(11):1665–1674. [DOI:10.1101/gr.6861907].

[75] Colella S, Yau C, Taylor JM, Mirza G, Butler H, Clouston P, Bassett AS, Seller A, Holmes CC, Ragoussis J: **QuantiSNP: an Objective Bayes Hidden-Markov Model to detect and accurately map copy number variation using SNP genotyping data**. *Nucleic Acids Research* 2007, **35**(6):2013–2025. [DOI:10.1093/nar/gkm076].

[76] Peiffer DA, Le JM, Steemers FJ, Chang W, Jenniges T, Garcia F, Haden K, Li J, Shaw CA, Belmont J, et al.: **High-resolution genomic profiling of chromosomal aberrations using Infinium whole-genome genotyping**. *Genome Research* 2006, **16**(9):1136–1148. [DOI:10.1101/gr.5402306].

[77] Valsesia A, Macé A, Jacquemont S, Beckmann JS, Kutalik Z: **The growing importance of CNVs: new insights for detection and clinical interpretation**. *Frontiers in Genetics* 2013, **4**(92). [DOI:10.3389/fgene.2013.00092].

[78] Cooper GM, Zerr T, Kidd JM, Eichler EE, Nickerson DA: **Systematic assessment of copy number variant detection via genome-wide SNP genotyping**. *Nature Genetics* 2008, **40**(10):1199–1203. [DOI:10.1038/ng.236].

[79] Sharp AJ, Locke DP, McGrath SD, Cheng Z, Bailey JA, Vallente RU, Pertz LM, Clark RA, Schwartz S, Segraves R, et al.: **Segmental duplications and copy-number variation in the human genome**. *American Journal of Human Genetics* 2005, **77**:78. [DOI:10.1086/431652].

[80] Kato M, Kawaguchi T, Ishikawa S, Umeda T, Nakamichi R, Shapero MH, Jones KW, Nakamura Y, Aburatani H, Tsunoda T: **Population-genetic nature of copy number variations in the human genome**. *Human Molecular Genetics* 2010, **19**(5):761–773. [DOI:10.1093/hmg/ddp541].

[81] Winchester L, Yau C, Ragoussis J: **Comparing CNV detection methods for SNP arrays**. *Briefings in Functional Genomics & Proteomics* 2009, **8**(5):353–366. [DOI:10.1093/bfgp/elp017].

[82] Eichler EE: **Widening the spectrum of human genetic variation**. *Nature Genetics* 2006, **38**:9–11. [DOI:10.1038/ng0106-9].

[83] Conrad DF, Andrews TD, Carter NP, Hurles ME, Pritchard JK: **A high-resolution survey of deletion polymorphism in the human genome**. *Nature Genetics* 2006, **38**:75–81. [DOI:10.1038/ng1697].

[84] McCarroll SA, Hadnott TN, Perry GH, Sabeti PC, Zody MC, Barrett JC, Dallaire S, Gabriel SB, Lee C, Daly MJ, et al.: **Common deletion polymorphisms in the human genome**. *Nature Genetics* 2006, **38**:86–92. [DOI:10.1038/ng1696].

[85] Medvedev P, Stanciu M, Brudno M: **Computational methods for discovering structural variation with next-generation sequencing**. *Nature methods* 2009, **6**:S13–S20. [DOI:10.1038/nmeth.1374].

[86] Kim UJ, Shizuya H, de Jong PJ, Birren B, Simon MI: **Stable propagation of cosmid sized human DNA inserts in an F factor based vector**. *Nucleic Acids Research* 1992, **20**(5):1083–1085. [DOI:10.1093/nar/20.5.1083].

[87] Tuzun E, Sharp AJ, Bailey JA, Kaul R, Morrison VA, Pertz LM, Haugen E, Hayden H, Albertson D, Pinkel D, et al.: **Fine-scale structural variation of the human genome**. *Nature Genetics* 2005, **37**(7):727–732. [DOI:10.1038/ng1562].

[88] Kidd JM, Cooper GM, Donahue WF, Hayden HS, Sampas N, Graves T, Hansen N, Teague B, Alkan C, Antonacci F, et al.: **Mapping and sequencing of structural variation from eight human genomes**. *Nature* 2008, **453**(7191):56–64. [DOI:10.1038/nature06862].

[89] Zhang Z, Schwartz S, Wagner L, Miller W: **A greedy algorithm for aligning DNA sequences**. *Journal of Computational Biology* 2000, **7**(1-2):203–214. [DOI:10.1089/10665270050081478].

[90] Li H, Durbin R: **Fast and accurate short read alignment with Burrows–Wheeler transform**. *Bioinformatics* 2009, **25**(14):1754–1760. [DOI:10.1093/bioinformatics/btp324].

[91] Li H, Durbin R: **Fast and accurate long-read alignment with Burrows–Wheeler transform**. *Bioinformatics* 2010, **26**(5):589–595. [DOI:10.1093/bioinformatics/btp698].

[92] Langmead B, Salzberg SL: **Fast gapped-read alignment with Bowtie 2**. *Nature Methods* 2012, **9**(4):357–359. [DOI:10.1038/nmeth.1923].

[93] Smith T, Waterman M: **Identification of Common Molecular Subsequences**. *Molecular Biology* 1981, **147**:195–197. [DOI:10.1016/0022-2836(81)90087-5].

[94] Park H, Kim JI, Ju YS, Gokcumen O, Mills RE, Kim S, Lee S, Suh D, Hong D, Kang HP, et al.: **Discovery of common Asian copy number variants using integrated high-resolution array CGH and massively parallel DNA sequencing**. *Nature Genetics* 2010, **42**(5):400–405. [DOI:10.1038/ng.555].

[95] Fromer M, Moran JL, Chambert K, Banks E, Bergen SE, Ruderfer DM, Handsaker RE, McCarroll SA, O'Donovan MC, Owen MJ, et al.: **Discovery and statistical genotyping of copy-number variation from whole-exome sequencing depth**. *The American Journal of Human Genetics* 2012, **91**(4):597–607. [DOI:10.1016/j.ajhg.2012.08.005].

[96] Komura D, Shen F, Ishikawa S, Fitch KR, Chen W, Zhang J, Liu G, Ihara S, Nakamura H, Hurles ME, et al.: **Genome-wide detection of human copy number variations using high-density DNA oligonucleotide arrays**. *Genome Research* 2006, **16**(12):1575–1584. [DOI:10.1101/gr.5629106].

[97] McCarroll SA, Kuruvilla FG, Korn JM, Cawley S, Nemesh J, Wysoker A, Shapero MH, de Bakker PI, Maller JB, Kirby A, et al.: **Integrated detection and population-genetic analysis of SNPs and copy number variation**. *Nature Genetics* 2008, **40**(10):1166–1174. [DOI:10.1038/ng.238].

[98] Bentley DR, Balasubramanian S, Swerdlow HP, Smith GP, Milton J, Brown CG, Hall KP, Evers DJ, Barnes CL, Bignell HR, et al.: **Accurate whole human genome sequencing using reversible terminator chemistry**. *Nature* 2008, **456**(7218):53–59. [DOI:10.1038/nature07517].

[99] The Wellcome Trust Case Control Consortium, Craddock N, Hurles ME, Cardin N, Pearson RD, Plagnol V, Robson S, Vukcevic D, Barnes C, Conrad DF, Giannoulatou E, et al.: **Genome-wide association study of CNVs in 16,000 cases of eight common diseases and 3,000 shared controls**. *Nature* 2010, **464**(7289):713–720. [DOI:10.1038/nature08979].

[100] Margulies M, Egholm M, Altman WE, Attiya S, Bader JS, Bemben LA, Berka J, Braverman MS, Chen YJ, Chen Z, et al.: **Genome sequencing in microfabricated high-density picolitre reactors**. *Nature* 2005, **437**(7057):376–380. [DOI:10.1038/nature03959].

[101] Ronaghi M: **Pyrosequencing sheds light on DNA sequencing**. *Genome Research* 2001, **11**:3–11. [DOI:10.1101/gr.150601].

[102] Mills RE, Walter K, Stewart C, Handsaker RE, Chen K, Alkan C, Abyzov A, Yoon SC, Ye K, Cheetham RK, et al.: **Mapping copy number variation by population-scale genome sequencing**. *Nature* 2011, **470**(7332):59–65. [DOI:10.1038/nature09708].

[103] Rimoin DL, Connor JM, Pyeritz RE, Korf BR: *Emery and Rimoin's principles and practice of medical genetics*. United States of America: Churchill Livingstone Elsevier, fifth edition 2007.

[104] Nussbaum RL, McInnes RR, Willard HR: *Thompson & Thompson Genetics in Medicine*. United States of America: Saunders, sixth edition 2004.

[105] Elston RC, Olson JM, Palmer L (Eds): *Biostatistical genetics and genetic epidemiology*. John Wiley & Sons Inc 2002.

[106] Ziegler A, König IR: *A statistical approach to genetic epidemiology: concepts and applications*. Vch Verlagsgesellschaft Mbh 2006.

[107] Pearson TA, Manolio TA: **How to interpret a genome-wide association study**. *The Journal of the American Medical Association* 2008, **299**(11):1335–1344. [DOI:10.1001/jama.299.11.1335].

[108] Clarke GM, Anderson CA, Pettersson FH, Cardon LR, Morris AP, Zondervan KT: **Basic statistical analysis in genetic case-control studies**. *Nature protocols* 2011, **6**(2):121–133. [DOI:10.1038/nprot.2010.182].

[109] Hindorff LA, MacArthur J, Morales J, Junkins HA, Hall PN, Klemm AK, Manolio TA: **A Catalog of Published Genome-Wide Association Studies** 2013, [www.genome.gov/gwastudies].

[110] Manolio TA, Collins FS, Cox NJ, Goldstein DB, Hindorff LA, Hunter DJ, McCarthy MI, Ramos EM, Cardon LR, Chakravarti A, et al.: **Finding the missing heritability of complex diseases**. *Nature* 2009, **461**(7265):747–753. [DOI:10.1038/nature08494].

[111] Maher B: **The case of the missing heritability**. *Nature* 2008, **456**(7218):18–21. [DOI:10.1038/456018a].

[112] Zuk O, Hechter E, Sunyaev SR, Lander ES: **The mystery of missing heritability: genetic interactions create phantom heritability**. *Proceedings of the National Academy of Sciences* 2012, **109**(4):1193–1198. [DOI:10.1073/pnas.1119675109].

[113] Eichler EE, Flint J, Gibson G, Kong A, Leal SM, Moore JH, Nadeau JH: **Missing heritability and strategies for finding the underlying causes of complex disease**. *Nature Reviews Genetics* 2010, **11**(6):446–450. [DOI:10.1038/nrg2809].

[114] Slatkin M: **Epigenetic inheritance and the missing heritability problem**. *Genetics* 2009, **182**(3):845–850. [DOI:10.1534/genetics.109.102798].

[115] Bell JT, Spector TD: **A twin approach to unraveling epigenetics**. *Trends in Genetics* 2011, **27**(3):116–125. [DOI:10.1016/j.tig.2010.12.005].

[116] Zhang B, Zhou Y, Lin N, Lowdon RF, Hong C, Nagarajan RP, Cheng JB, Li D, Stevens M, Lee HJ, et al.: **Functional DNA methylation differences between tissues, cell types, and across individuals discovered using the M&M algorithm**. *Genome Research* 2013, **23**(9):1522–1540. [DOI:10.1101/gr.156539.113].

[117] Moskvina V, Craddock N, Holmans P, Owen MJ, O'Donovan MC: **Effects of differential genotyping error rate on the type I error probability of case-control studies**. *Human Heredity* 2006, **61**:55–64. [DOI:10.1159/000092553].

[118] Lee SH, Wray NR, Goddard ME, Visscher PM: **Estimating missing heritability for disease from genome-wide association studies**. *The American Journal of Human Genetics* 2011, **88**(3):294–305. [DOI:10.1016/j.ajhg.2011.02.002].

[119] Chanock SJ, Manolio T, Boehnke M, Boerwinkle E, Hunter DJ, Thomas G, Hirschhorn JN, Abecasis G, Altshuler D, Bailey-Wilson JE, et al.: **Replicating genotype–phenotype associations**. *Nature* 2007, **447**(7145):655–660. [DOI:10.1038/447655a].

[120] The Wellcome Trust Case Control Consortium: **Genome-wide association study of 14,000 cases of seven common diseases and 3,000 shared controls**. *Nature* 2007, **447**(7145):661–678. [DOI:10.1038/nature05911].

[121] Gibson G: **Rare and common variants: twenty arguments**. *Nature Reviews Genetics* 2012, **13**(2):135–145. [DOI:10.1038/nrg3118].

[122] Jouan L, Gauthier J, Dion PA, Rouleau GA: **Rare variants in complex traits: novel identification strategies and the role of de novo mutations**. *Human Heredity* 2012, **74**(3-4):215–225. [DOI:10.1159/000346478].

[123] Tennessen JA, Bigham AW, O'Connor TD, Fu W, Kenny EE, Gravel S, McGee S, Do R, Liu X, Jun G, et al.: **Evolution and functional impact of rare coding variation from deep sequencing of human exomes**. *Science* 2012, **337**(6090):64–69. [DOI:10.1126/science.1219240].

[124] Nelson MR, Wegmann D, Ehm MG, Kessner D, Jean PS, Verzilli C, Shen J, Tang Z, Bacanu SA, Fraser D, et al.: **An abundance of rare functional variants in 202 drug target genes sequenced in 14,002 people**. *Science* 2012, **337**(6090):100–104. [DOI:10.1126/science.1217876].

[125] Schork NJ, Murray SS, Frazer KA, Topol EJ: **Common vs. rare allele hypotheses for complex diseases**. *Current Opinion in Genetics & Development* 2009, **19**(3):212–219. [DOI:10.1016/j.gde.2009.04.010].

[126] Tardif JC, Rhéaume E, Lemieux Perreault LP, Grégoire JC, Feroz Zada Y, Asselin G, Provost S, Barhdadi A, Rhainds D, L'Allier PL, Ibrahim R, Upmanyu R, Niesor EJ, Benghozi R, Suchankova G, Laghrissi-Thode F, Guertin MC, Olsson AG, Mongrain I, Schwartz GG, Dubé MP: **Pharmacogenomic Determinants of the Cardiovascular Effects of Dalcetrapib**. *The New England Journal of Medicine* 2013. [Submitted].

[127] Turner S, Armstrong LL, Bradford Y, Carlson CS, Crawford DC, Crenshaw AT, Andrade M, Doheny KF, Haines JL, Hayes G, et al.: **Quality Control Procedures for Genome-Wide Association Studies**. *Current Protocols in Human Genetics* 2011, :1–19. [DOI:10.1002/0471142905.hg0119s68].

[128] Purcell S, Neale B, Todd-Brown K, Thomas L, Ferreira MA, Bender D, Maller J, Sklar P, De Bakker PI, Daly MJ, et al.: **PLINK: a tool set for whole-genome association and population-based linkage analyses**. *The American Journal of Human Genetics* 2007, **81**(3):559–575. [DOI:10.1086/519795].

[129] Anderson CA, Pettersson FH, Clarke GM, Cardon LR, Morris AP, Zondervan KT: **Data quality control in genetic case-control association studies**. *Nature Protocols* 2010, **5**(9):1564–1573. [DOI:10.1038/nprot.2010.116].

[130] Laurie CC, Doheny KF, Mirel DB, Pugh EW, Bierut LJ, Bhangale T, Boehm F, Caporaso NE, Cornelis MC, Edenberg HJ, et al.: **Quality control and quality assurance in genotypic data for genome-wide association studies**. *Genetic Epidemiology* 2010, **34**(6):591–602. [DOI:10.1002/gepi.20516].

[131] Stevens EL, Heckenberg G, Roberson ED, Baugher JD, Downey TJ, Pevsner J: **Inference of relationships in population data using identity-by-descent and identity-by-state**. *PLoS Genetics* 2011, **7**(9):e1002287. [DOI:10.1371/journal.pgen.1002287].

[132] Kaiser J: **Genetic Influences on Disease Remain Hidden**. *Science* 2012, **338**(6110):1016–1017. [DOI:10.1126/science.338.6110.1016].

[133] llumina Inc: **Data Sheet: Human Exome BeadChips** 2013, [http://res.illumina.com/documents/products/datasheets/datasheet_humanexome_beadchips.pdf].

[134] Ritchie ME, Liu R, Carvalho BS, Irizarry RA, et al.: **Comparing genotyping algorithms for Illumina's Infinium whole-genome SNP BeadChips**. *BMC Bioinformatics* 2011, **12**:68. [DOI:10.1186/1471-2105-12-68].

[135] Giannoulatou E, Yau C, Colella S, Ragoussis J, Holmes CC: **GenoSNP: a variational Bayes within-sample SNP genotyping algorithm that does not require a reference population**. *Bioinformatics* 2008, **24**(19):2209–2214. [DOI:10.1093/bioinformatics/btn386].

[136] Li G, Gelernter J, Kranzler HR, Zhao H: **M3: an improved SNP calling algorithm for Illumina BeadArray data**. *Bioinformatics* 2012, **28**(3):358–365.

[137] Shah T, Liu J, Floyd J, Morris J, Wirth N, Barrett J, Anderson C: **optiCall: a robust genotype-calling algorithm for rare, low-frequency and common variants**. *Bioinformatics* 2012, **28**(12):1598–1603. [DOI:10.1093/bioinformatics/bts180].

[138] llumina Inc: **GenomeStudio Software v2011.1 Release Notes** 2011, [http://support.illumina.com/documents/MyIllumina/1f2d00ae-b759-489b-82f9-0163ce085ae7/GenomeStudio_2011.1_Release_Notes.pdf].

[139] Goldstein JI, Crenshaw A, Carey J, Grant GB, Maguire J, Fromer M, O'Dushlaine C, Moran JL, Chambert K, Stevens C, et al.: **zCall: a rare variant caller for array-based genotyping Genetics and population analysis**. *Bioinformatics* 2012, **28**(19):2543–2545. [DOI:10.1093/bioinformatics/bts479].

[140] Arthur D, Vassilvitskii S: **k-means++: the advantages of careful seeding**. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '07, Philadelphia, PA, USA: Society for Industrial and Applied Mathematics 2007:1027–1035, [http://dl.acm.org/citation.cfm?id=1283383.1283494].

[141] NHLBI GO Exome Sequencing Project (ESP): **Exome Variant Server** 2013, [http://evs.gs.washington.edu/EVS/].

[142] Gwet KL: **Computing inter-rater reliability and its variance in the presence of high agreement**. *British Journal of Mathematical and Statistical Psychology* 2008, **61**:29–48. [DOI:10.1348/000711006X126600].

[143] Fleiss JL: **Measuring nominal scale agreement among many raters**. *Psychological Bulletin* 1971, **76**(5):378–382.

[144] Lemieux Perreault LP, Provost S, Legault MA, Barhdadi A, Dubé MP: **pyGenClean: efficient tool for genetic data clean up before association testing**. *Bioinformatics* 2013, **29**(13):1704–1705. [DOI:10.1093/bioinformatics/btt261].

[145] Powers S, Gopalakrishnan S, Tintle N: **Assessing the impact of non-differential genotyping errors on rare variant tests of association**. *Human Heredity* 2011, **72**(3):153–160. [DOI:10.1159/000332222].

[146] Mayer-Jochimsen M, Fast S, Tintle NL: **Assessing the impact of differential genotyping errors on rare variant tests of association**. *PloS One* 2013, **8**(3):e56626. [DOI:10.1371/journal.pone.0056626].

[147] Stranger BE, Forrest MS, Dunning M, Ingle CE, Beazley C, Thorne N, Redon R, Bird CP, De Grassi A, Lee C, et al.: **Relative impact of nucleotide and copy number variation on gene expression phenotypes**. *Science* 2007, **315**(5813):848–853. [DOI:10.1126/science.1136678].

[148] Clerget-Darpoux F, Elston RC: **Are linkage analysis and the collection of family data dead? Prospects for family studies in the age of genome-wide association**. *Human Heredity* 2007, **64**(2):91–96. [DOI:10.1159/000101960].

[149] Elston RC, Danyu L, Gang Z: **Multistage Sampling for Genetic Studies**. *Annual Review of Genomics and Human Genetics* 2007, **8**:327–342. [DOI:10.1146/annurev.genom.8.080706.092357].

[150] Nagelkerke NJ, Hoebee B, Teunis P, Kimman TG: **Combining the transmission disequilibrium test and case–control methodology using generalized logistic regression**. *European Journal of Human Genetics* 2004, **12**(11):964–970. [DOI:10.1038/sj.ejhg.5201255].

[151] Wang T, Elston RC: **A quantitative linkage score for an association study following a linkage analysis**. *BMC Genetics* 2006, **7**:5–16. [DOI:10.1186/1471-2156-7-5].

[152] Abecasis GR, Cherny SS, Cookson WO, Cardon LR: **Merlin—rapid analysis of dense genetic maps using sparse gene flow trees**. *Nature Genetics* 2002, **30**:97–101. [DOI:10.1038/ng786].

[153] O'Connell JR, Weeks DE: **PedCheck: a program for identification of genotype incompatibilities in linkage analysis**. *The American Journal of Human Genetics* 1998, **63**:259–266. [DOI:10.1086/301904].

[154] Kirov G, Grozeva D, Norton N, Ivanov D, Mantripragada KK, Holmans P, Craddock N, Owen MJ, O'Donovan MC, et al.: **Support for the involvement of large copy number variants in the pathogenesis of schizophrenia**. *Human Molecular Genetics* 2009, **18**(8):1497–1503. [DOI:10.1093/hmg/ddp043].

[155] Stefansson H, Rujescu D, Cichon S, Pietiläinen OP, Ingason A, Steinberg S, Fossdal R, Sigurdsson E, Sigmundsson T, Buizer-Voskamp JE, et al.: **Large recurrent microdeletions associated with schizophrenia**. *nature* 2008, **455**(7210):232–236. [DOI:10.1038/nature07229].

[156] McCarthy SE, Makarov V, Kirov G, Addington AM, McClellan J, Yoon S, Perkins DO, Dickel DE, Kusenda M, Krastoshevsky O, et al.: **Microduplications of 16p11.2 are associated with schizophrenia**. *Nature genetics* 2009, **41**(11):1223–1227. [DOI:10.1038/ng.474].

[157] Ingason A, Rujescu D, Cichon S, Sigurdsson E, Sigmundsson T, Pietiläinen O, Buizer-Voskamp J, Strengman E, Francks C, Muglia P, et al.: **Copy number variations of chromosome 16p13.1 region associated with schizophrenia**. *Molecular Psychiatry* 2011, **16**:17–25. [DOI:10.1038/mp.2009.101].

[158] Glessner JT, Reilly MP, Kim CE, Takahashi N, Albano A, Hou C, Bradfield JP, Zhang H, Sleiman PM, Flory JH, et al.: **Strong synaptic transmission impact by copy number variations in schizophrenia**. *Proceedings of the National Academy of Sciences* 2010, **107**(23):10584–10589. [DOI:10.1073/pnas.1000274107].

[159] Bochukova EG, Huang N, Keogh J, Henning E, Purmann C, Blaszczyk K, Saeed S, Hamilton-Shield J, Clayton-Smith J, O'Rahilly S, et al.: **Large, rare chromosomal deletions associated with severe early-onset obesity**. *Nature* 2010, **463**(7281):666–670. [DOI:10.1038/nature08689].

[160] Heinzen EL, Radtke RA, Urban TJ, Cavalleri GL, Depondt C, Need AC, Walley NM, Nicoletti P, Ge D, Catarino CB, et al.: **Rare deletions at 16p13.11 predispose to a diverse spectrum of sporadic epilepsy syndromes**. *The American Journal of Human Genetics* 2010, **86**(5):707–718. [DOI:10.1016/j.ajhg.2010.03.018].

[161] Glessner JT, Wang K, Cai G, Korvatska O, Kim CE, Wood S, Zhang H, Estes A, Brune CW, Bradfield JP, et al.: **Autism genome-wide copy number variation reveals ubiquitin and neuronal genes**. *Nature* 2009, **459**(7246):569–573. [DOI:10.1038/nature07953].

[162] Pinto D, Pagnamenta AT, Klei L, Anney R, Merico D, Regan R, Conroy J, Magalhaes TR, Correia C, Abrahams BS, et al.: **Functional impact of global rare copy number variation in autism spectrum disorders**. *Nature* 2010, **466**(7304):368–372. [DOI:10.1038/nature09146].

[163] Zhang D, Cheng L, Qian Y, Alliey-Rodriguez N, Kelsoe JR, Greenwood T, Nievergelt C, Barrett TB, McKinney R, Schork N, et al.: **Singleton deletions throughout the genome increase risk of bipolar disorder**. *Molecular Psychiatry* 2009, **14**(4):376–380. [DOI:10.1038/mp.2008.144].

[164] Williams NM, Zaharieva I, Martin A, Langley K, Mantripragada K, Fossdal R, Stefansson H, Stefansson K, Magnusson P, Gudmundsson OO, et al.: **Rare chromosomal deletions and duplications in attention-deficit hyperactivity disorder: a genome-wide analysis**. *The Lancet* 2010, **376**(9750):1401–1408. [DOI:10.1016/S0140-6736(10)61109-9].

[165] Walsh T, McClellan JM, McCarthy SE, Addington AM, Pierce SB, Cooper GM, Nord AS, Kusenda M, Malhotra D, Bhandari A, et al.: **Rare structural variants disrupt multiple genes in neurodevelopmental pathways in schizophrenia**. *Science* 2008, **320**(5875):539–543. [DOI:10.1126/science.1155174].

[166] Vrijenhoek T, Buizer-Voskamp JE, van der Stelt I, Strengman E, Sabatti C, Geurts van Kessel A, Brunner HG, Ophoff RA, Veltman JA: **Recurrent CNVs disrupt three candidate genes in schizophrenia patients**. *The American Journal of Human Genetics* 2008, **83**(4):504–510. [DOI:10.1016/j.ajhg.2008.09.011].

[167] Xu B, Roos JL, Levy S, Van Rensburg E, Gogos JA, Karayiorgou M: **Strong association of de novo copy number mutations with sporadic schizophrenia**. *Nature Genetics* 2008, **40**(7):880–885. [DOI:10.1038/ng.162].

[168] International Schizophrenia Consortium: **Rare chromosomal deletions and duplications increase risk of schizophrenia**. *Nature* 2008, **455**(7210):237–241. [DOI:10.1038/nature07239].

[169] Hirschhorn JN, Daly MJ: **Genome-wide association studies for common diseases and complex traits**. *Nature Reviews Genetics* 2005, **6**(2):95–108. [DOI:10.1038/nrg1521].

[170] Wang WY, Barratt BJ, Clayton DG, Todd JA: **Genome-wide association studies: theoretical and practical concerns**. *Nature Reviews Genetics* 2005, **6**(2):109–118. [DOI:10.1038/nrg1522].

[171] McCarthy MI, Abecasis GR, Cardon LR, Goldstein DB, Little J, Ioannidis JP, Hirschhorn JN: **Genome-wide association studies for complex traits: consensus, uncertainty and challenges**. *Nature Reviews Genetics* 2008, **9**(5):356–369. [DOI:10.1002/cyto.990190104].

[172] Scherer SW, Lee C, Birney E, Altshuler DM, Eichler EE, Carter NP, Hurles ME, Feuk L: **Challenges and standards in integrating surveys of structural variation**. *Nature Genetics* 2007, **39**(7 Suppl):S7–S15. [DOI:10.1038/ng2093].

[173] Perry GH, Ben-Dor A, Tsalenko A, Sampas N, Rodriguez-Revenga L, Tran CW, Scheffer A, Steinfeld I, Tsang P, Yamada NA, et al.: **The fine-scale and complex architecture of human copy-number variation**. *The American Journal of Human Genetics* 2008, **82**(3):685–695. [DOI:10.1016/j.ajhg.2007.12.010].

[174] McCarroll SA: **Extending genome-wide association studies to copy-number variation**. *Human Molecular Genetics* 2008, **17**(R2):R135–R142. [DOI:10.1093/hmg/ddn282].

[175] McCarroll SA, Altshuler DM: **Copy-number variation and association studies of human disease**. *Nature Genetics* 2007, **39**(7 Suppl):S37–S42. [DOI:10.1038/ng2080].

[176] de Vries B, Pfundt R, Leisink M, Koolen DA, Vissers LE, Janssen IM, Reijmersdal Sv, Nillesen WM, Huys EH, Leeuw Nd, et al.: **Diagnostic genome profiling in mental retardation**. *The American Journal of Human Genetics* 2005, **77**(4):606–616. [DOI:10.1086/491719].

[177] Myocardial Infarction Genetics Consortium: **Genome-wide association of early-onset myocardial infarction with single nucleotide polymorphisms and copy number variants**. *Nature Genetics* 2009, **41**(3):334–341. [DOI:10.1038/ng.327].

[178] Pinto D, Darvishi K, Shi X, Rajan D, Rigler D, Fitzgerald T, Lionel AC, Thiruvahindrapuram B, MacDonald JR, Mills R, et al.: **Comprehensive assessment of array-based platforms and calling algorithms for detection of copy number variants**. *Nature Biotechnology* 2011, **29**(6):512–520. [DOI:10.1038/nbt.1852].

[179] Curtis C, Lynch A, Dunning M, Spiteri I, Marioni J, Hadfield J, Chin SF, Brenton J, Tavaré S, Caldas C: **The pitfalls of platform comparison: DNA copy number array technologies assessed**. *BMC Genomics* 2009, **10**:588. [DOI:10.1186/1471-2164-10-588].

[180] Halper-Stromberg E, Frelin L, Ruczinski I, Scharpf R, Jie C, Carvalho B, Hao H, Hetrick K, Jedlicka A, Dziedzic A, et al.: **Performance assessment of copy number microarray platforms using a spike-in experiment**. *Bioinformatics* 2011, **27**(8):1052–1060. [DOI:10.1093/bioinformatics/btr106].

[181] Dellinger AE, Saw SM, Goh LK, Seielstad M, Young TL, Li YJ: **Comparative analyses of seven algorithms for copy number variant identification from single nucleotide polymorphism arrays**. *Nucleic Acids Research* 2010, **38**(9):e105–e105. [DOI:10.1093/nar/gkq040].

[182] Tsuang DW, Millard SP, Ely B, Chi P, Wang K, Raskind WH, Kim S, Brkanac Z, Yu CE: **The effect of algorithms on copy number variant detection**. *PloS One* 2010, **5**(12):e14456. [DOI:10.1371/journal.pone.0014456].

[183] Zhang D, Qian Y, Akula N, Alliey-Rodriguez N, Tang J, Gershon ES, Liu C, et al.: **Accuracy of CNV detection from GWAS data**. *PLoS One* 2011, **6**:e14511. [DOI:10.1371/journal.pone.0014511].

[184] Olivier M: **A haplotype map of the human genome**. *Physiological Genomics* 2003, **13**:3–9. [DOI:10.1152/physiolgenomics.00178.2002].

[185] Barrett JC, Lee JC, Lees CW, Prescott NJ, Anderson CA, Phillips A, Wesley E, Parnell K, Zhang H, Drummond H, et al.: **Genome-wide association study of ulcerative colitis identifies three new susceptibility loci, including the HNF4A region**. *Nature Genetics* 2009, **41**(12):1330–1334. [DOI:10.1038/ng.483].

[186] Spencer CC, Plagnol V, Strange A, Gardner M, Paisan-Ruiz C, Band G, Barker RA, Bellenguez C, Bhatia K, Blackburn H, et al.: **Dissection of the genetics of Parkinson's disease identifies an additional association 5' of SNCA and multiple associated haplotypes at 17q21**. *Human Molecular Genetics* 2011, **20**(2):345–353. [DOI:10.1093/hmg/ddq469].

[187] Strange A, Capon F, Spencer C, Knight J, Weale ME, Allen MH, Barton A, Band G, Bellenguez C, Bergboer J, et al.: **A genome-wide association study identifies new psoriasis susceptibility loci and an interaction between HLA-C and ERAP1**. *Nature Genetics* 2010, **42**(11):985–990. [DOI:10.1038/ng.694].

[188] Walters R, Jacquemont S, Valsesia A, De Smith A, Martinet D, Andersson J, Falchi M, Chen F, Andrieux J, Lobbens S, et al.: **A new highly penetrant form of obesity due to deletions on chromosome 16p11.2**. *Nature* 2010, **463**(7281):671–675. [DOI:10.1038/nature08727].

[189] Fakhro KA, Choi M, Ware SM, Belmont JW, Towbin JA, Lifton RP, Khokha MK, Brueckner M: **Rare copy number variations in congenital heart disease patients identify unique genes in left-right patterning**. *Proceedings of the National Academy of Sciences* 2011, **108**(7):2915–2920. [DOI:10.1073/pnas.1019645108].

[190] Blauw HM, Veldink JH, van Es MA, van Vught PW, Saris CG, van der Zwaag B, Franke L, Burbach JPH, Wokke JH, Ophoff RA, et al.: **Copy-number variation in sporadic amyotrophic lateral sclerosis: a genome-wide screen**. *The Lancet Neurology* 2008, **7**(4):319–326. [DOI:10.1016/S1474-4422(08)70048-6].

[191] Wang K, Zhang H, Bloss C, Duvvuri V, Kaye W, Schork N, Berrettini W, Hakonarson H: **A genome-wide association study on common SNPs and rare CNVs in anorexia nervosa**. *Molecular Psychiatry* 2011, **16**(9):949–959. [DOI:10.1038/mp.2010.107].

[192] Yang TL, Chen XD, Guo Y, Lei SF, Wang JT, Zhou Q, Pan F, Chen Y, Zhang ZX, Dong SS, et al.: **Genome-wide Copy-Number-Variation Study Identified a Susceptibility Gene, UGT2B17, for Osteoporosis**. *The American Journal of Human Genetics* 2008, **83**(6):663–674. [DOI:10.1016/j.ajhg.2008.10.006].

[193] Itsara A, Cooper GM, Baker C, Girirajan S, Li J, Absher D, Krauss RM, Myers RM, Ridker PM, Chasman DI, et al.: **Population analysis of large copy number variants and hotspots of human genetic disease**. *The American Journal of Human Genetics* 2009, **84**(2):148–161. [DOI:10.1016/j.ajhg.2008.12.014].

[194] Pinto D, Marshall C, Feuk L, Scherer SW: **Copy-number variation in control population cohorts**. *Human Molecular Genetics* 2007, **16**(R2):R168–R173. [DOI:10.1093/hmg/ddm241].

[195] Al-Sukhni W, Gallinger S: **Germline copy number variation in control populations**. *Cytogenetic and Genome Research* 2008, **123**(1-4):211–223. [DOI:10.1159/000184711].

[196] Glessner JT, Bradfield JP, Wang K, Takahashi N, Zhang H, Sleiman PM, Mentch FD, Kim CE, Hou C, Thomas KA, et al.: **A genome-wide study reveals copy number variants exclusive to childhood obesity cases**. *The American Journal of Human Genetics* 2010, **87**(5):661–666. [DOI:10.1016/j.ajhg.2010.09.014].

[197] Greenway SC, Pereira AC, Lin JC, DePalma SR, Israel SJ, Mesquita SM, Ergul E, Conta JH, Korn JM, McCarroll SA, et al.: **De novo copy number variants identify new genes and loci in isolated sporadic tetralogy of Fallot**. *Nature Genetics* 2009, **41**(8):931–935. [DOI:10.1038/ng.415].

[198] Pearson RD: **Bias due to selection of rare variants using frequency in controls**. *Nature Genetics* 2011, **43**(5):392–393. [DOI:10.1038/ng.816].

[199] Marshall CR, Noor A, Vincent JB, Lionel AC, Feuk L, Skaug J, Shago M, Moessner R, Pinto D, Ren Y, et al.: **Structural variation of chromosomes in autism spectrum disorder**. *The American Journal of Human Genetics* 2008, **82**(2):477–488. [DOI:10.1016/j.ajhg.2007.12.009].

[200] Zöllner S, Teslovich TM: **Using GWAS data to identify copy number variants contributing to common complex diseases**. *Statistical Science* 2009, **24**(4):530–546. [DOI:10.1214/09-STS304].

[201] Raychaudhuri S, Korn JM, McCarroll SA, Altshuler D, Sklar P, Purcell S, Daly MJ, et al.: **Accurately assessing the risk of schizophrenia conferred by rare copy-number variation affecting genes with brain function**. *PLoS Genetics* 2010, **6**(9):e1001097. [DOI:10.1371/journal.pgen.1001097].

[202] Girard SL, Gauthier J, Noreau A, Xiong L, Zhou S, Jouan L, Dionne-Laporte A, Spiegelman D, Henrion E, Diallo O, et al.: **Increased exonic de novo mutation rate in individuals with schizophrenia**. *Nature Genetics* 2011, **43**(9):860–863. [DOI:10.1038/ng.886].

[203] O'Roak BJ, Deriziotis P, Lee C, Vives L, Schwartz JJ, Girirajan S, Karakoc E, MacKenzie AP, Ng SB, Baker C, et al.: **Exome sequencing in sporadic autism spectrum disorders identifies severe de novo mutations**. *Nature Genetics* 2011, **43**(6):585–589. [DOI:10.1038/ng.835].

[204] Xu B, Roos JL, Dexheimer P, Boone B, Plummer B, Levy S, Gogos JA, Karayiorgou M: **Exome sequencing supports a de novo mutational paradigm for schizophrenia**. *Nature Genetics* 2011, **43**(9):864–868. [DOI:10.1038/ng.902].

Annexe I
Comparison of genotype clustering tools with rare variants
Additional Materials

Comparison of genotype clustering tools with rare variants

Additional Materials

Lemieux Perreault L.-P., Legault M.-A., Barhdadi A., Provost S., Normand V., Tardif J.-C. and Dubé M.-P.

Supplemental Equation 1 - Error rate for rare markers

The genotypic model for error rate estimation was tested by Liu *et al.* for common variants only. However, we found that the possible values of ϵ were out of bound (*i.e.* negative or above one) for a majority of rare markers. For those cases, ϵ was approximated using $\epsilon \simeq (C_1 - C_3 + 1)/3$, as described below.

$$
\begin{aligned}
C_1 &= p_1^2(1-2\epsilon) + 2p_1p_2\epsilon + p_2^2\epsilon && \text{(S1)}\\
C_3 &= p_1^2\epsilon + 2p_1p_2\epsilon + p_2^2(1-2\epsilon) && \text{(S2)}\\
C_1 - C_3 &= p_1^2(1-2\epsilon) + 2p_1p_2\epsilon + p_2^2\epsilon - p_1^2\epsilon - 2p_1p_2\epsilon - p_2^2(1-2\epsilon)\\
&= p_1^2(1-2\epsilon) + p_2^2\epsilon - p_1^2\epsilon - p_2^2(1-2\epsilon)\\
&= p_1^2 - 2p_1^2\epsilon + p_2^2\epsilon - p_1^2\epsilon - p_2^2 + 2p_2^2\epsilon\\
&= p_1^2 - 3p_1^2\epsilon + 3p_2^2\epsilon - p_2^2\\
&= (p_1^2 - p_2^2) - 3(p_1^2 - p_2^2)\epsilon\\
&= (1-3\epsilon)(p_1^2 - p_2^2)\\
&= (1-3\epsilon)(p_1 - p_2)(p_1 + p_2)\\
&= (1-3\epsilon)(p_1 - (1-p_1))\\
C_1 - C_3 &= (1-3\epsilon)(2p_1 - 1) && \text{(S3)}\\
2p_1 - 1 &= \frac{C_1 - C_3}{1-3\epsilon}\\
2p_1 &= \frac{C_1 - C_3}{1-3\epsilon} + 1\\
p_1 &= \frac{1}{2}\left(\frac{C_1 - C_3}{1-3\epsilon}\right) + \frac{1}{2} && \text{(S4)}\\
\text{if } p_1 \approx 0 &\Rightarrow \frac{1}{2}\left(\frac{C_1 - C_3}{1-3\epsilon}\right) + \frac{1}{2} \approx 0\\
&\Rightarrow \frac{C_1 - C_3}{1-3\epsilon} + 1 \approx 0\\
&\Rightarrow C_1 - C_3 + 1 - 3\epsilon \approx 0\\
&\Rightarrow C_1 - C_3 + 1 \approx 3\epsilon\\
&\Rightarrow \epsilon \approx \frac{C_1 - C_3 + 1}{3} && \text{(S5)}
\end{aligned}
$$

Supplemental Table 1 - Overall agreement probability and Cohen's κ calculation

Table S1: Overall agreement probability and Cohen's κ calculation. Distribution of n samples by calling tool in q categories. The set of possible categories are all possible genotypes (*i.e.* $q \in \{\text{AA}, \text{AB}, \text{BB}, 00\}$, where 00 represents the *no call* category). This table is computed for each marker and for each pair of calling tools. The overall agreement probability and Cohen's κ are shown in Equation 1 and 2 of the main text, respectively.

	Tool B				
Tool A	1	2	$\cdots$	q	*Total*
1	n_{11}	n_{12}	$\cdots$	n_{1q}	n_{A1}
2	n_{21}	n_{22}	$\cdots$	n_{2q}	n_{A2}
$\vdots$			$\cdots$		$\vdots$
q	n_{q1}	n_{q2}	$\cdots$	n_{qq}	n_{Aq}
Total	n_{B1}	n_{B2}	$\cdots$	n_{Bq}	n

Supplemental Table 2 - Fleiss' π calculation

Table S2: Fleiss' π calculation. Distribution of r calling tools by n samples and q response categories. The set of possible categories are all possible genotypes (*i.e.* $q \in \{\text{AA}, \text{AB}, \text{BB}, 00\}$, where 00 represents the *no call* category). This table is computed for each marker and for each calling tool. Fleiss' π is explained in Equation 3 of the main text.

	Category				
Sample	1	2	$\cdots$	q	*Total*
1	r_{11}	r_{12}	$\cdots$	r_{1q}	r
2	r_{21}	r_{22}	$\cdots$	r_{2q}	r
$\vdots$			$\cdots$		$\vdots$
n	r_{n1}	r_{n2}	$\cdots$	r_{nq}	r
Total	r_{+1}	r_{+2}	$\cdots$	r_{+q}	nr

Supplemental Table 3 - Call concordance with the 1000 Genomes Project (Fleiss' π outliers)

Table S3: Call concordance with the *1000 Genomes Project* (Fleiss's π outliers). Call concordance and number of compared markers for the three control replicates when compared to the *1000 Genomes Project* for the markers that were outliers for their Fleiss' π values. The following four tools were compared: *GenCall* (optimized cluster file), *GenoSNP* (optimized), *optiCall* (without excluding markers failing Hardy-Weinberg) and *zCall*.

Tool	NA12763_R		NA12763_R1		NA12763_R2	
	Rate	Number	Rate	Number	Rate	Number
GenCall (optimized)	0.989157	3,228	0.989151	3,226	0.989434	3,218
GenoSNP (optimized)	0.895096	3,079	0.908626	3,130	0.878186	3,021
optiCall	0.851575	3,207	0.849688	3,200	0.830272	3,158
zCall	0.984485	3,416	0.984485	3,416	0.984485	3,416

Annexe II
Methodological Challenges in Genome-wide Association Studies of Rare Copy Number Variants Supplementary Figures

Supplementary Figures

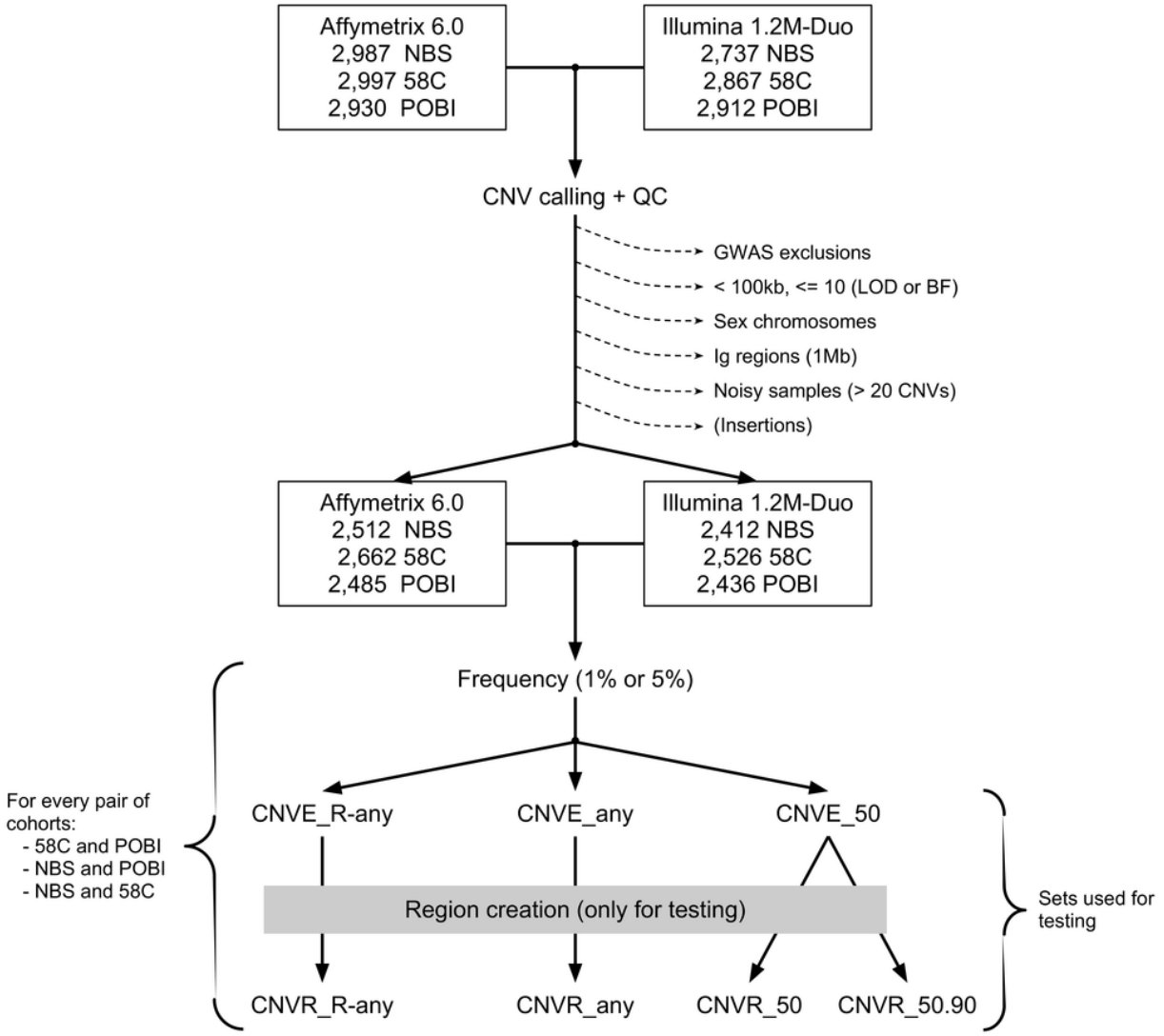


Figure S1: Analysis pipeline. A diagram showing the analysis pipeline, from raw data (both Affymetrix and Illumina) to the final sets of CNVs..

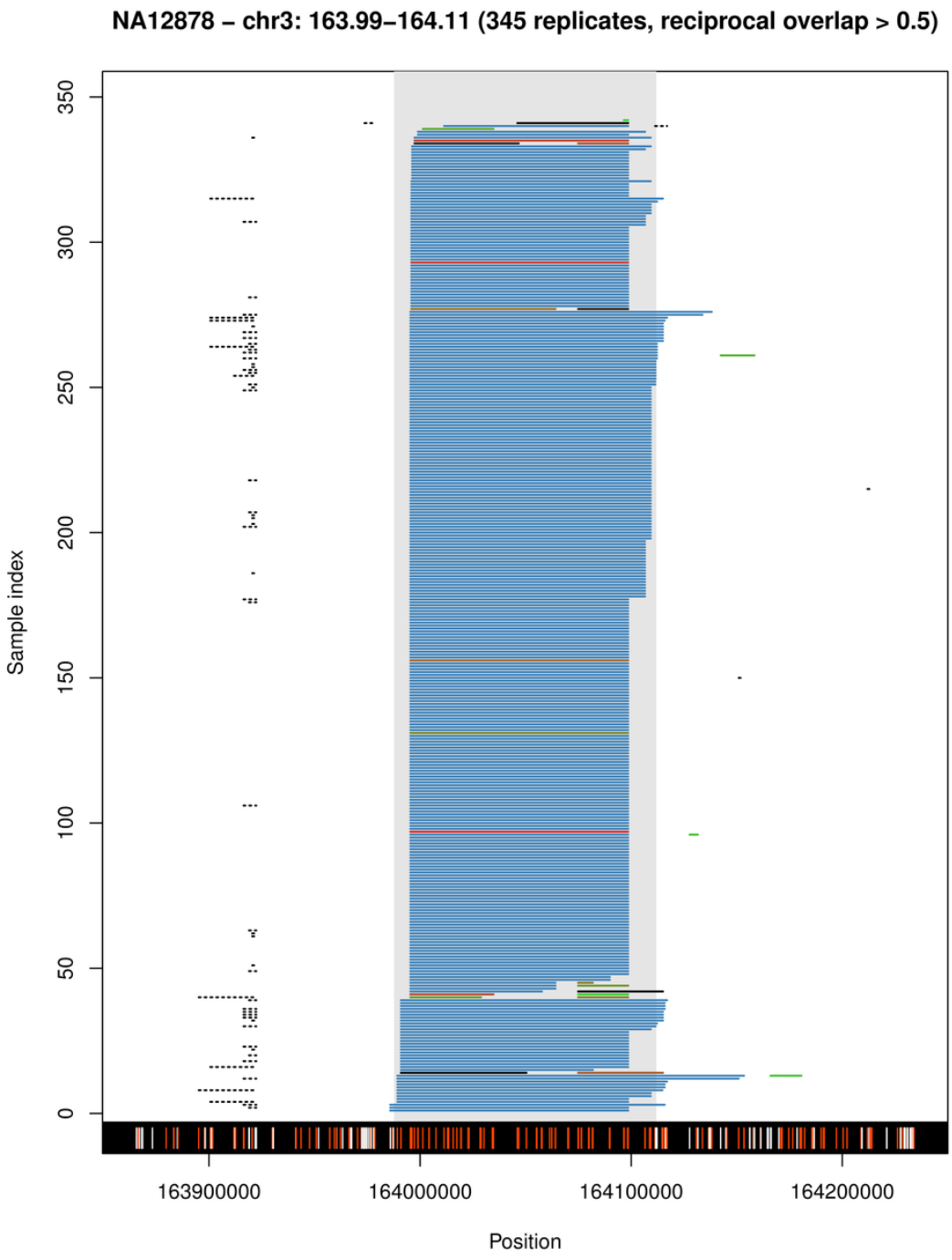


Figure S2: NA12878's raw CNV calls in known 1000 Genomes deletions (Affymetrix). Blue lines represent deletions that pass the confidence threshold (LOD score > 10) and the reciprocal overlap threshold (> 50%). Black lines are those that do not pass the reciprocal overlap threshold (<= 50%), but still over the confidence threshold. Dotted lines represent duplications. A range of color, from green to red, represent lower or higher values of LOD score, respectively, but still below the threshold. The black band represents the different probes in that region, where white and orange ticks represent SNP and CN probes, respectively. The gray rectangle shows the exact region of the 1000 Genomes Project deletions.

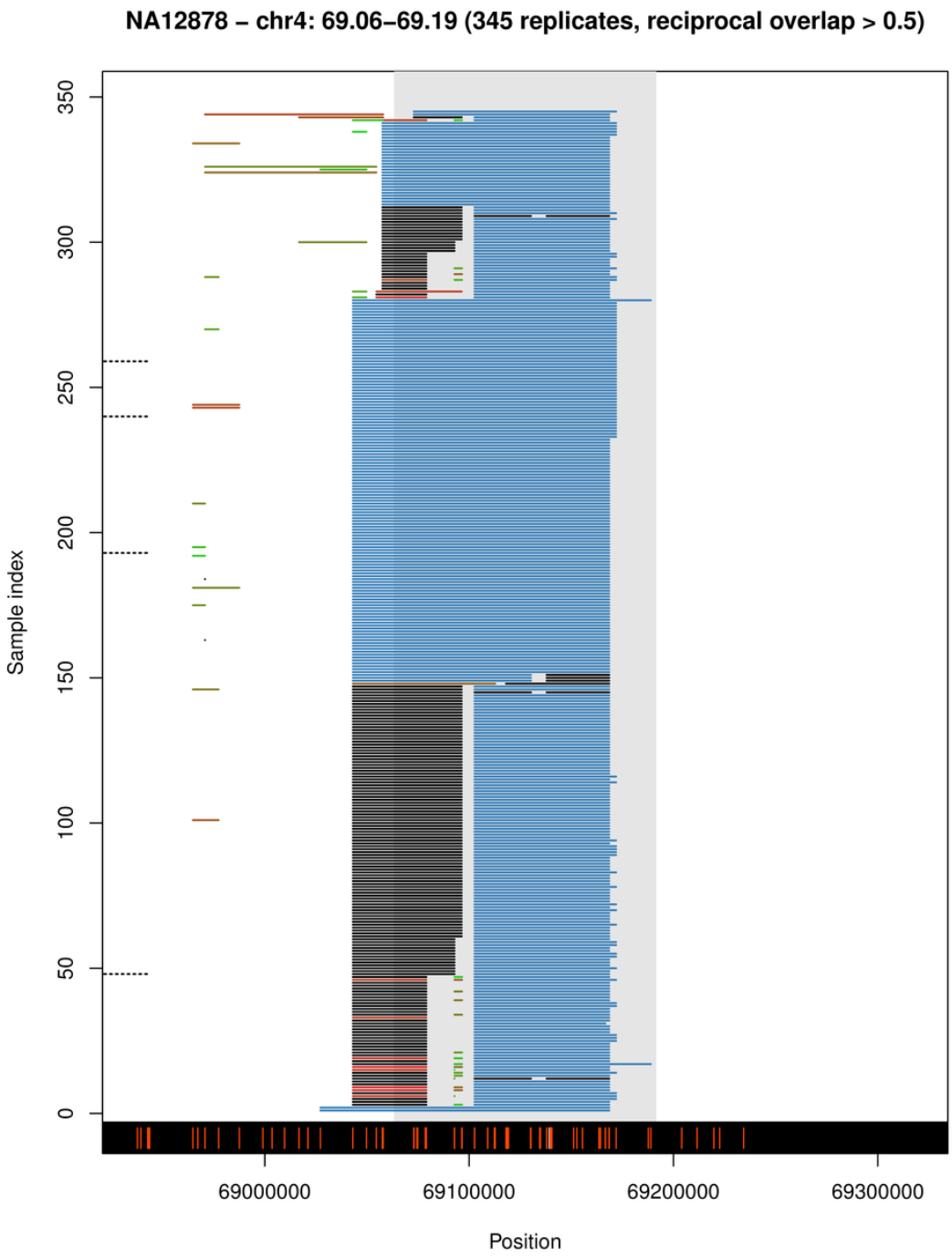


Figure S3: NA12878's raw CNV calls in known 1000 Genomes deletions (Affymetrix). Blue lines represent deletions that pass the confidence threshold (LOD score > 10) and the reciprocal overlap threshold (> 50%). Black lines are those that do not pass the reciprocal overlap threshold (<= 50%), but still over the confidence threshold. Dotted lines represent duplications. A range of color, from green to red, represent lower or higher values of LOD score, respectively, but still below the threshold. The black band represents the different probes in that region, where white and orange ticks represent SNP and CN probes, respectively. The gray rectangle shows the exact region of the 1000 Genomes Project deletions.

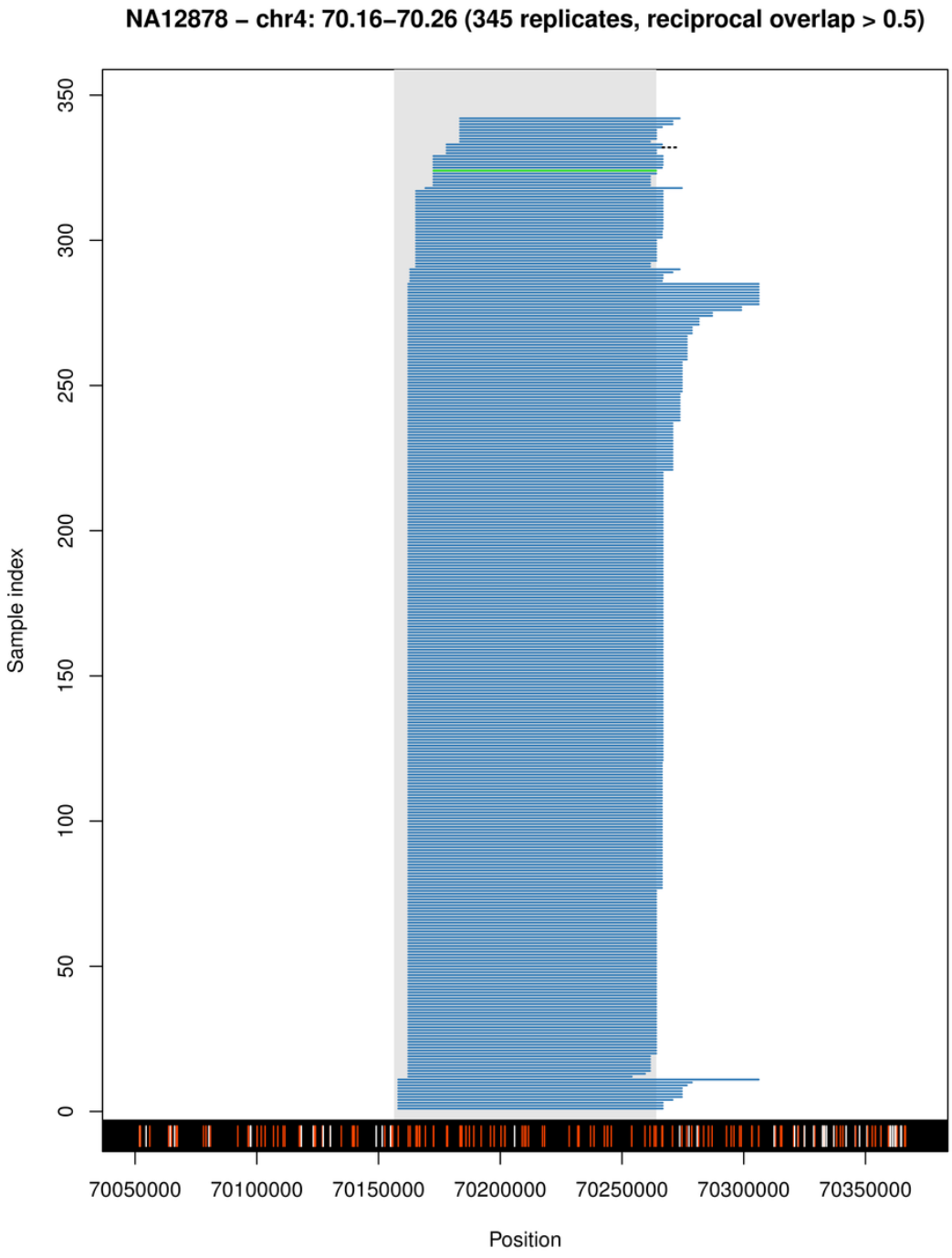


Figure S4: NA12878's raw CNV calls in known 1000 Genomes deletions (Affymetrix). Blue lines represent deletions that pass the confidence threshold (LOD score > 10) and the reciprocal overlap threshold (> 50%). Black lines are those that do not pass the reciprocal overlap threshold (<= 50%), but still over the confidence threshold. Dotted lines represent duplications. A range of color, from green to red, represent lower or higher values of LOD score, respectively, but still below the threshold. The black band represents the different probes in that region, where white and orange ticks represent SNP and CN probes, respectively. The gray rectangle shows the exact region of the 1000 Genomes Project deletions.

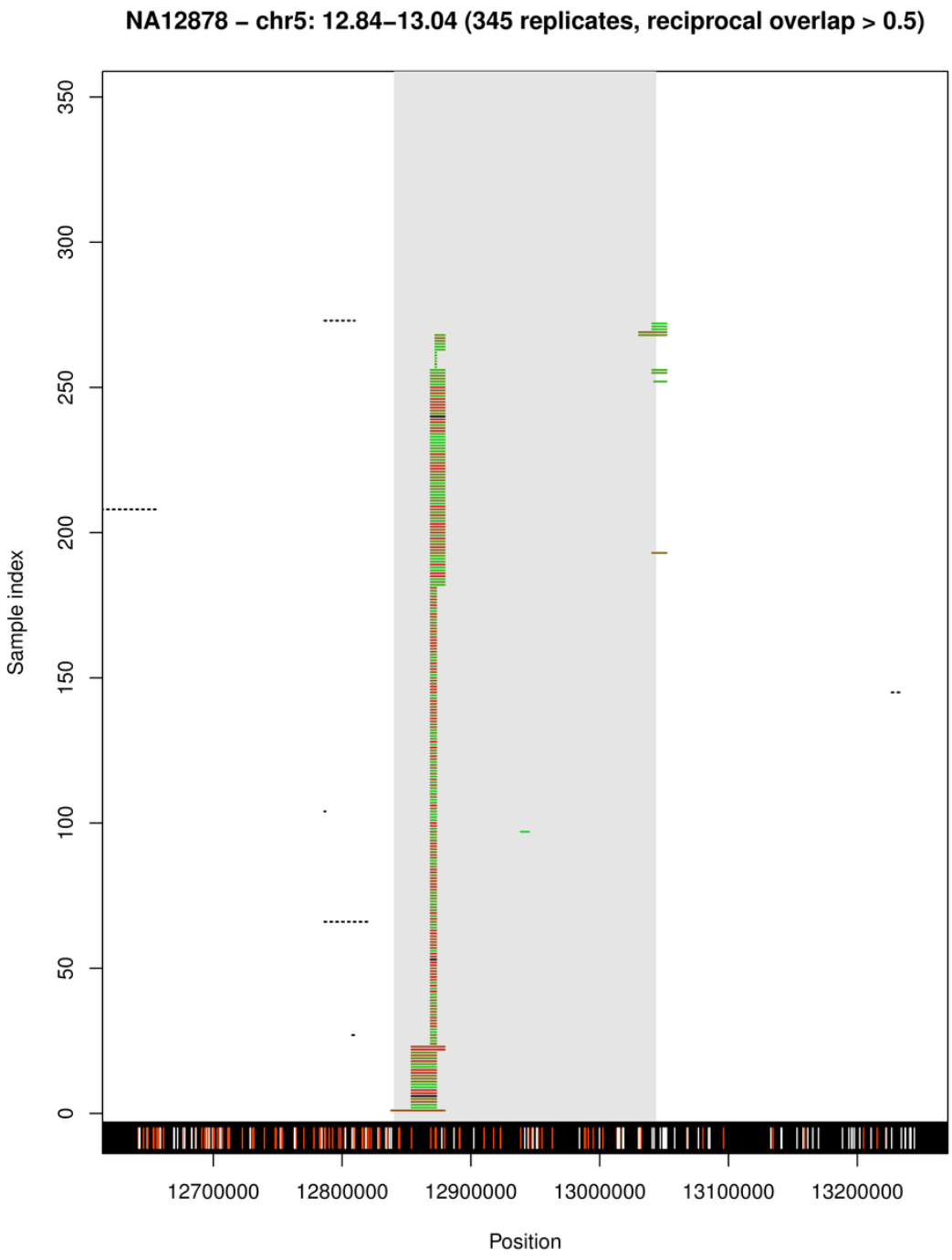


Figure S5: NA12878's raw CNV calls in known 1000 Genomes deletions (Affymetrix). Blue lines represent deletions that pass the confidence threshold (LOD score > 10) and the reciprocal overlap threshold (> 50%). Black lines are those that do not pass the reciprocal overlap threshold (<= 50%), but still over the confidence threshold. Dotted lines represent duplications. A range of color, from green to red, represent lower or higher values of LOD score, respectively, but still below the threshold. The black band represents the different probes in that region, where white and orange ticks represent SNP and CN probes, respectively. The gray rectangle shows the exact region of the 1000 Genomes Project deletions.

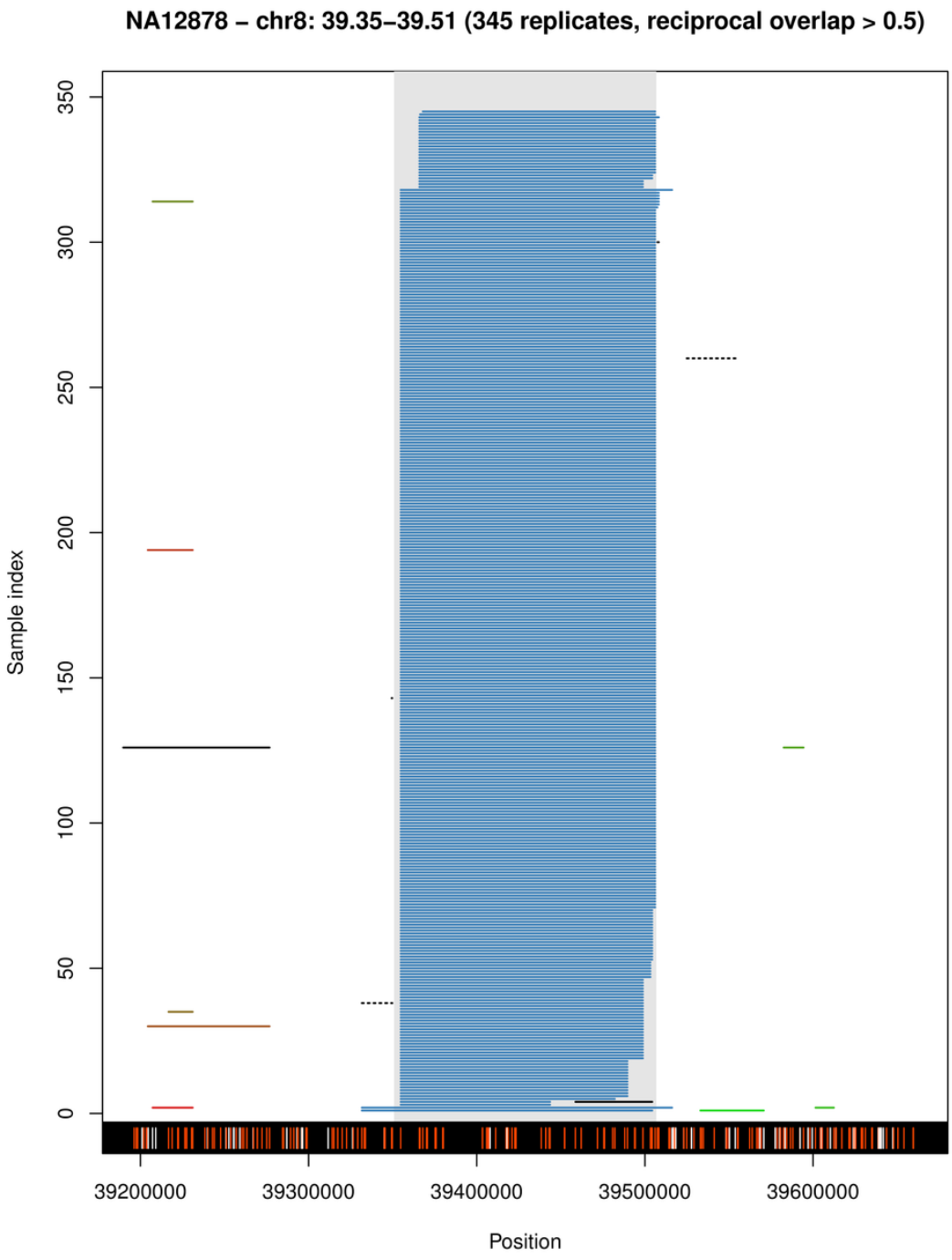


Figure S6: NA12878's raw CNV calls in known 1000 Genomes deletions (Affymetrix). Blue lines represent deletions that pass the confidence threshold (LOD score > 10) and the reciprocal overlap threshold (> 50%). Black lines are those that do not pass the reciprocal overlap threshold (<= 50%), but still over the confidence threshold. Dotted lines represent duplications. A range of color, from green to red, represent lower or higher values of LOD score, respectively, but still below the threshold. The black band represents the different probes in that region, where white and orange ticks represent SNP and CN probes, respectively. The gray rectangle shows the exact region of the 1000 Genomes Project deletions.

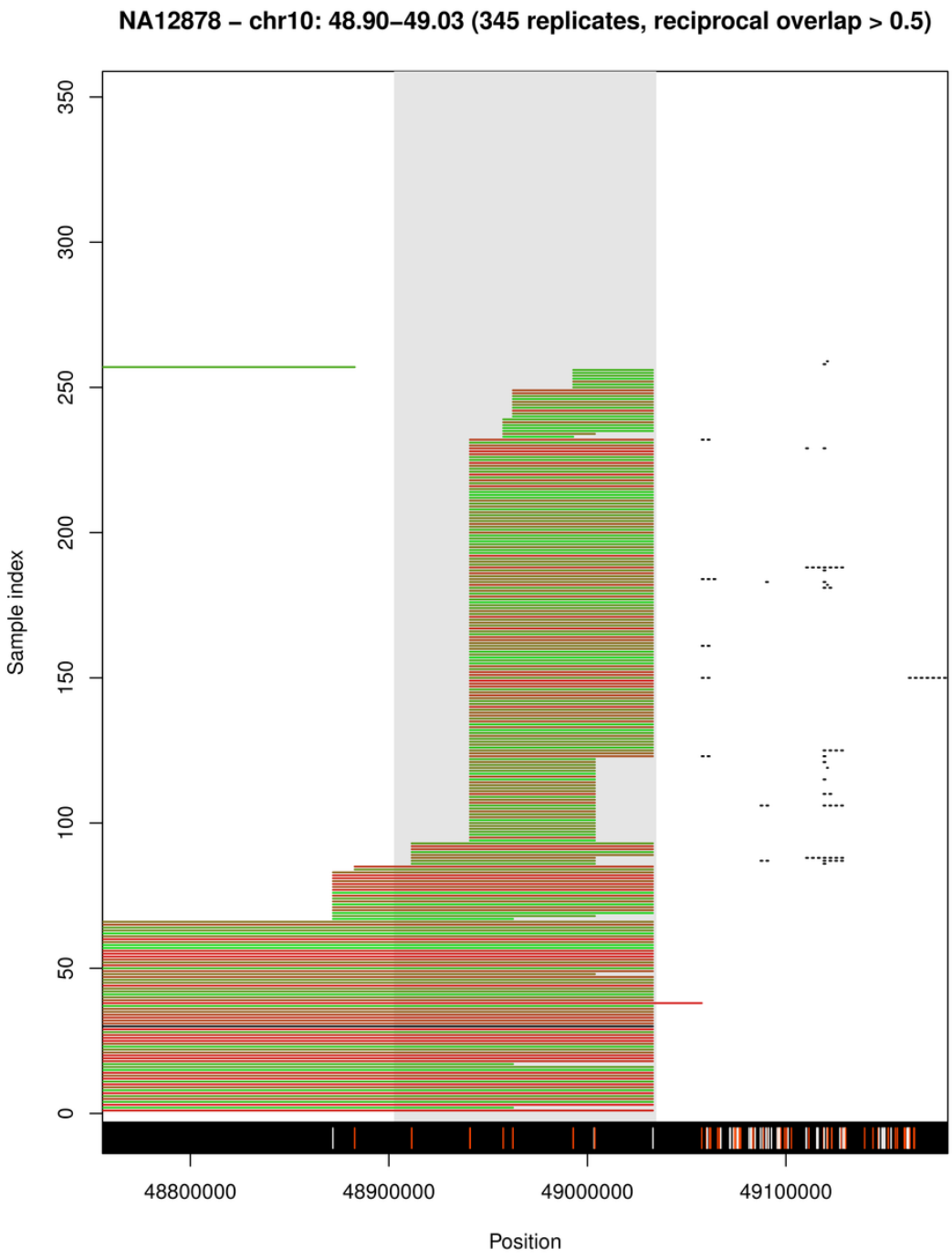


Figure S7: NA12878's raw CNV calls in known 1000 Genomes deletions (Affymetrix). Blue lines represent deletions that pass the confidence threshold (LOD score > 10) and the reciprocal overlap threshold (> 50%). Black lines are those that do not pass the reciprocal overlap threshold (<= 50%), but still over the confidence threshold. Dotted lines represent duplications. A range of color, from green to red, represent lower or higher values of LOD score, respectively, but still below the threshold. The black band represents the different probes in that region, where white and orange ticks represent SNP and CN probes, respectively. The gray rectangle shows the exact region of the 1000 Genomes Project deletions.

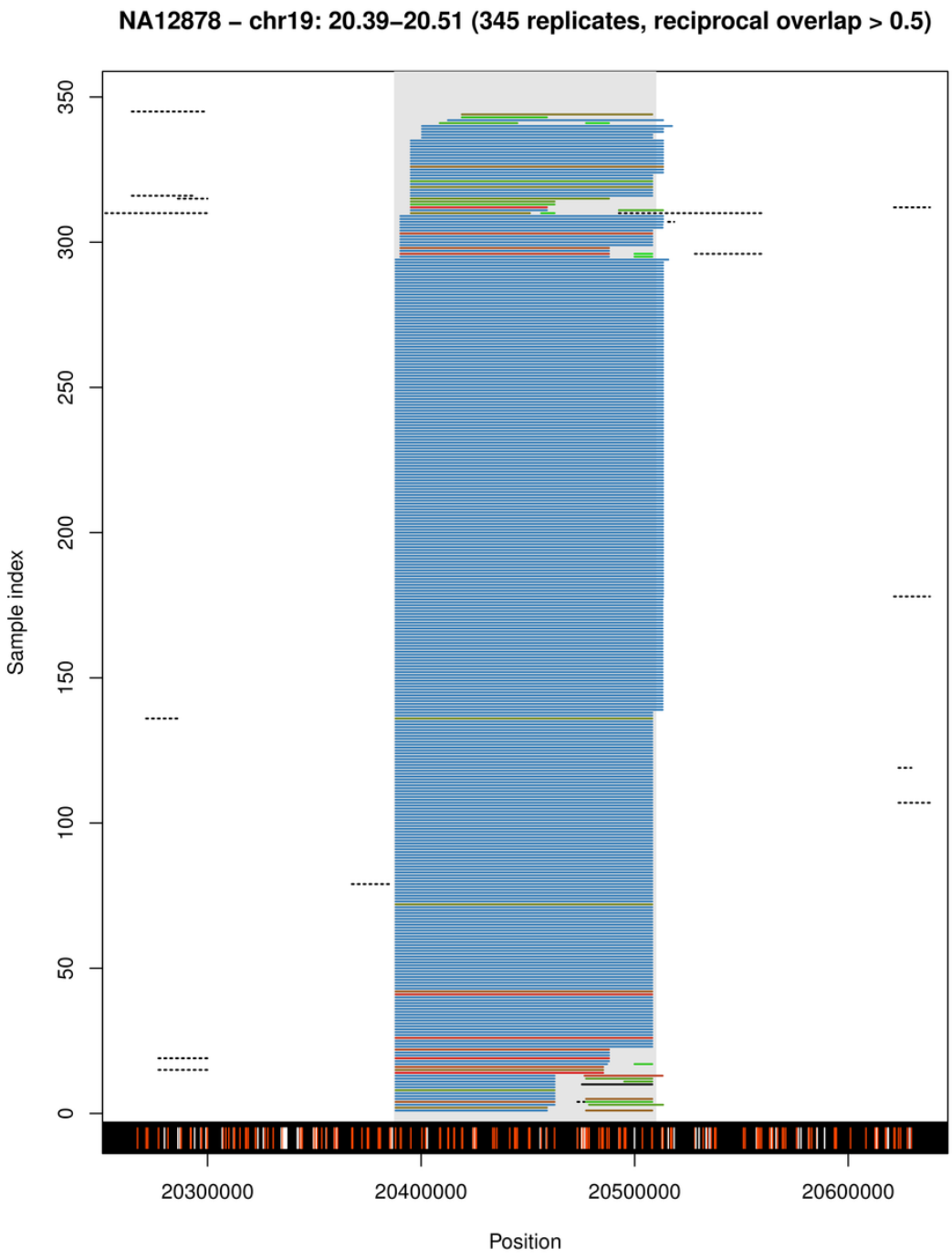


Figure S8: NA12878's raw CNV calls in known 1000 Genomes deletions (Affymetrix). Blue lines represent deletions that pass the confidence threshold (LOD score > 10) and the reciprocal overlap threshold (> 50%). Black lines are those that do not pass the reciprocal overlap threshold (<= 50%), but still over the confidence threshold. Dotted lines represent duplications. A range of color, from green to red, represent lower or higher values of LOD score, respectively, but still below the threshold. The black band represents the different probes in that region, where white and orange ticks represent SNP and CN probes, respectively. The gray rectangle shows the exact region of the 1000 Genomes Project deletions.

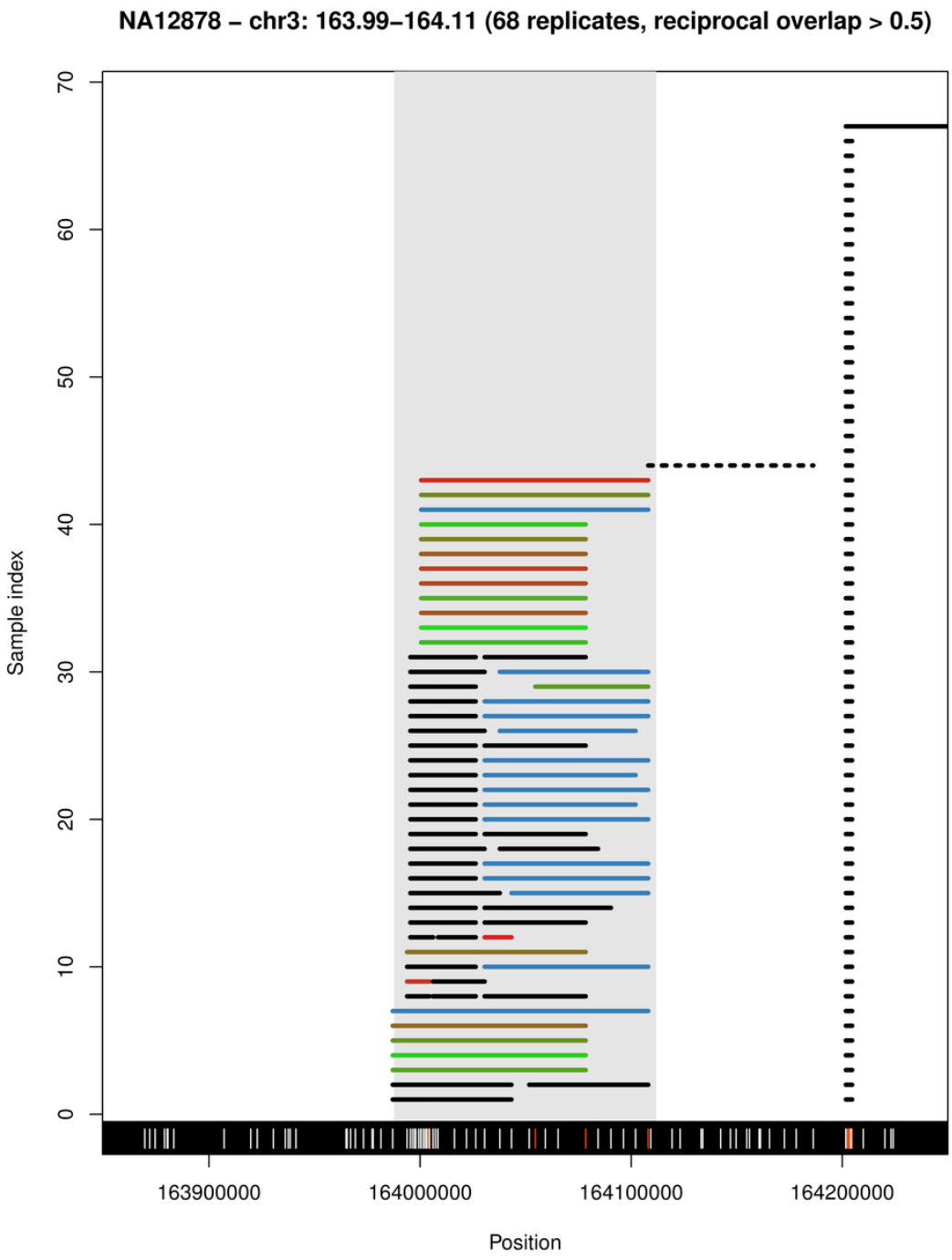


Figure S9: NA12878's raw CNV calls in known 1000 Genomes deletions (Illumina). Blue lines represent deletions that pass the confidence threshold (BF > 10) and the reciprocal overlap threshold (> 50%). Black lines are those that do not pass the reciprocal overlap threshold (<= 50%), but still over the confidence threshold. Dotted lines represent duplications. A range of color, from green to red, represent lower or higher values of BF, respectively, but still below the threshold. The black band at the bottom represents the different probes in that region, where white and orange ticks represent SNP and CN probes, respectively. The gray rectangle shows the exact region of the 1000 Genomes Project deletions.

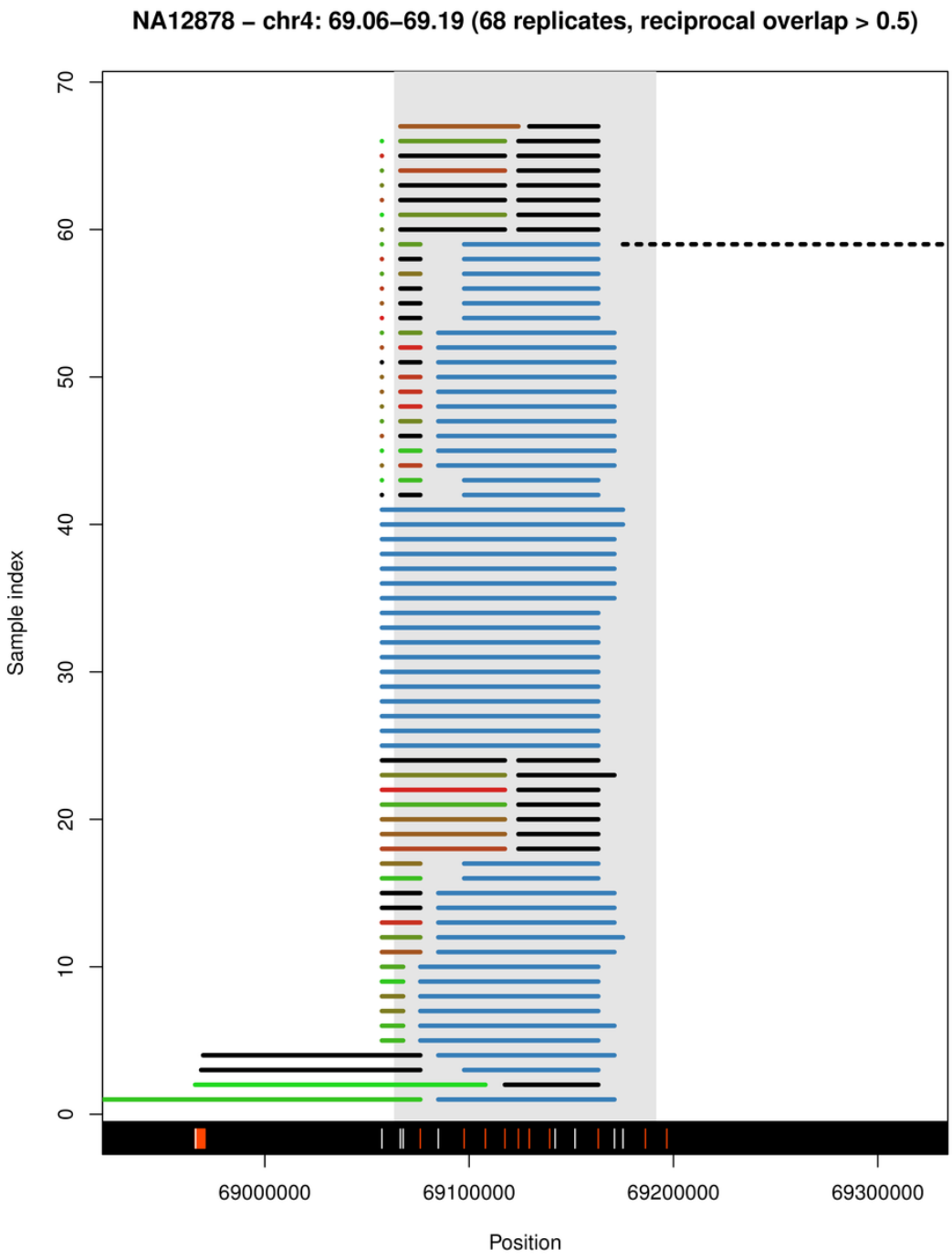


Figure S10: NA12878's raw CNV calls in known 1000 Genomes deletions (Illumina). Blue lines represent deletions that pass the confidence threshold (BF > 10) and the reciprocal overlap threshold (> 50%). Black lines are those that do not pass the reciprocal overlap threshold (<= 50%), but still over the confidence threshold. Dotted lines represent duplications. A range of color, from green to red, represent lower or higher values of BF, respectively, but still below the threshold. The black band at the bottom represents the different probes in that region, where white and orange ticks represent SNP and CN probes, respectively. The gray rectangle shows the exact region of the 1000 Genomes Project deletions.

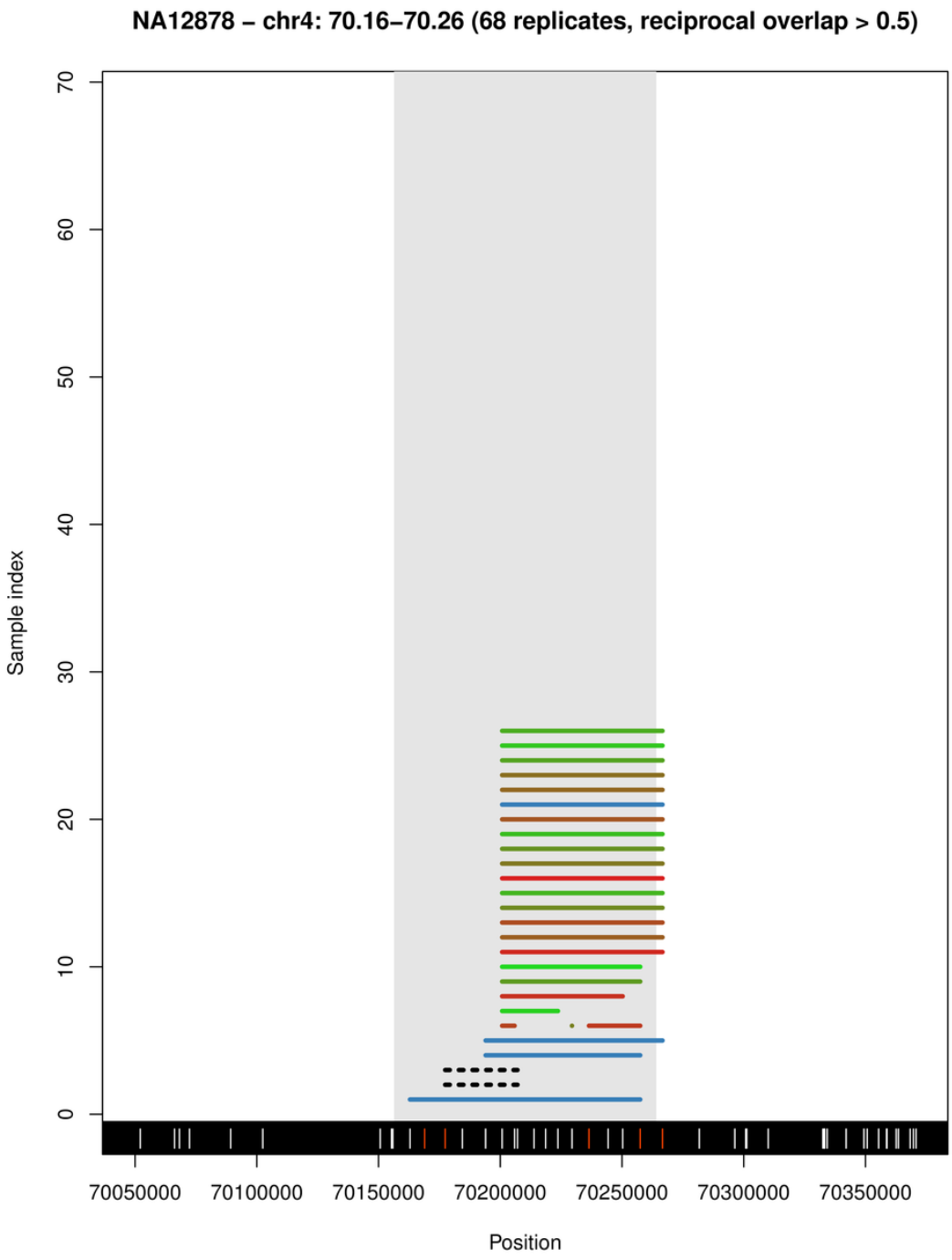


Figure S11: NA12878's raw CNV calls in known 1000 Genomes deletions (Illumina). Blue lines represent deletions that pass the confidence threshold (BF > 10) and the reciprocal overlap threshold (> 50%). Black lines are those that do not pass the reciprocal overlap threshold (<= 50%), but still over the confidence threshold. Dotted lines represent duplications. A range of color, from green to red, represent lower or higher values of BF, respectively, but still below the threshold. The black band at the bottom represents the different probes in that region, where white and orange ticks represent SNP and CN probes, respectively. The gray rectangle shows the exact region of the 1000 Genomes Project deletions.

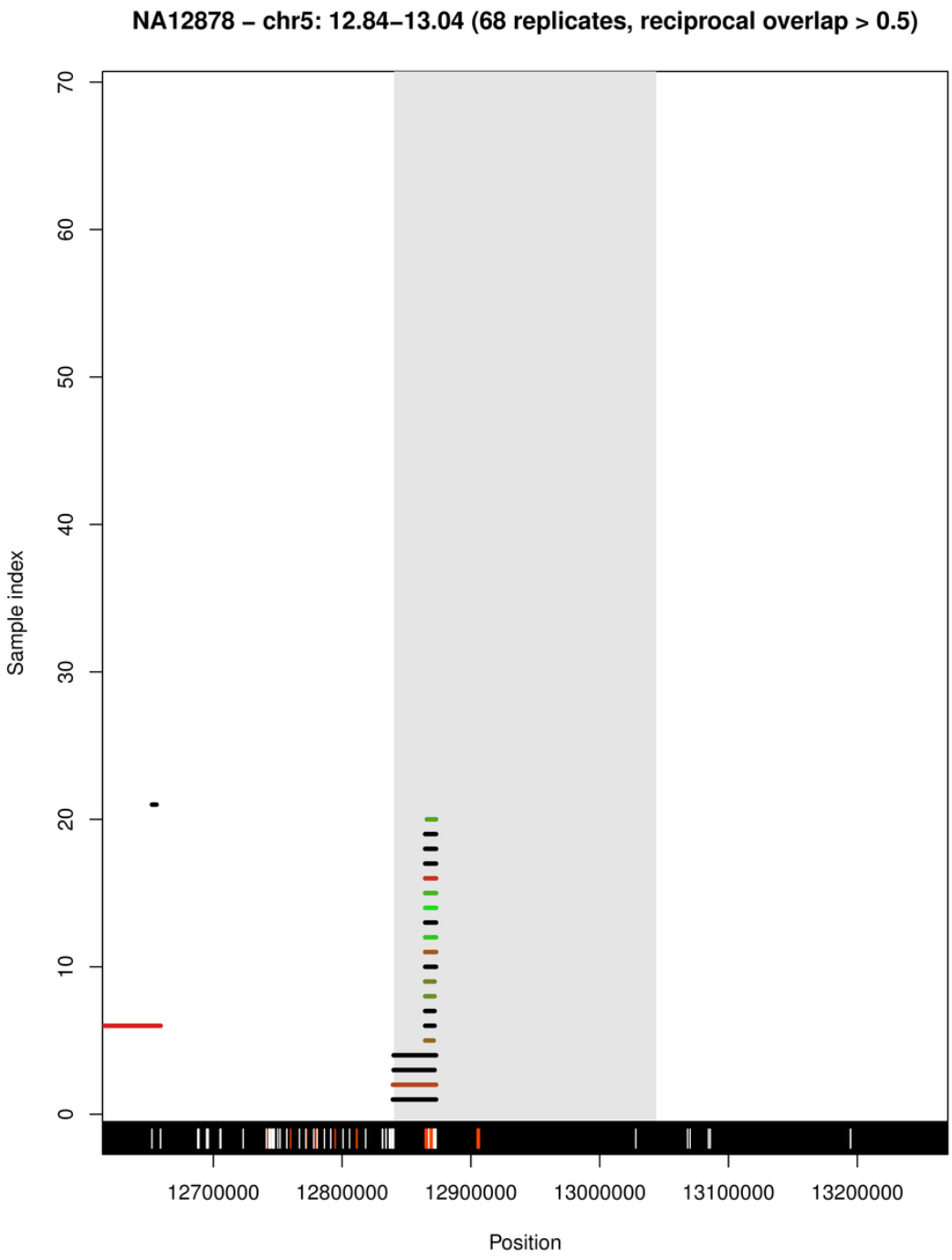


Figure S12: NA12878's raw CNV calls in known 1000 Genomes deletions (Illumina). Blue lines represent deletions that pass the confidence threshold (BF > 10) and the reciprocal overlap threshold (> 50%). Black lines are those that do not pass the reciprocal overlap threshold (<= 50%), but still over the confidence threshold. Dotted lines represent duplications. A range of color, from green to red, represent lower or higher values of BF, respectively, but still below the threshold. The black band at the bottom represents the different probes in that region, where white and orange ticks represent SNP and CN probes, respectively. The gray rectangle shows the exact region of the 1000 Genomes Project deletions.

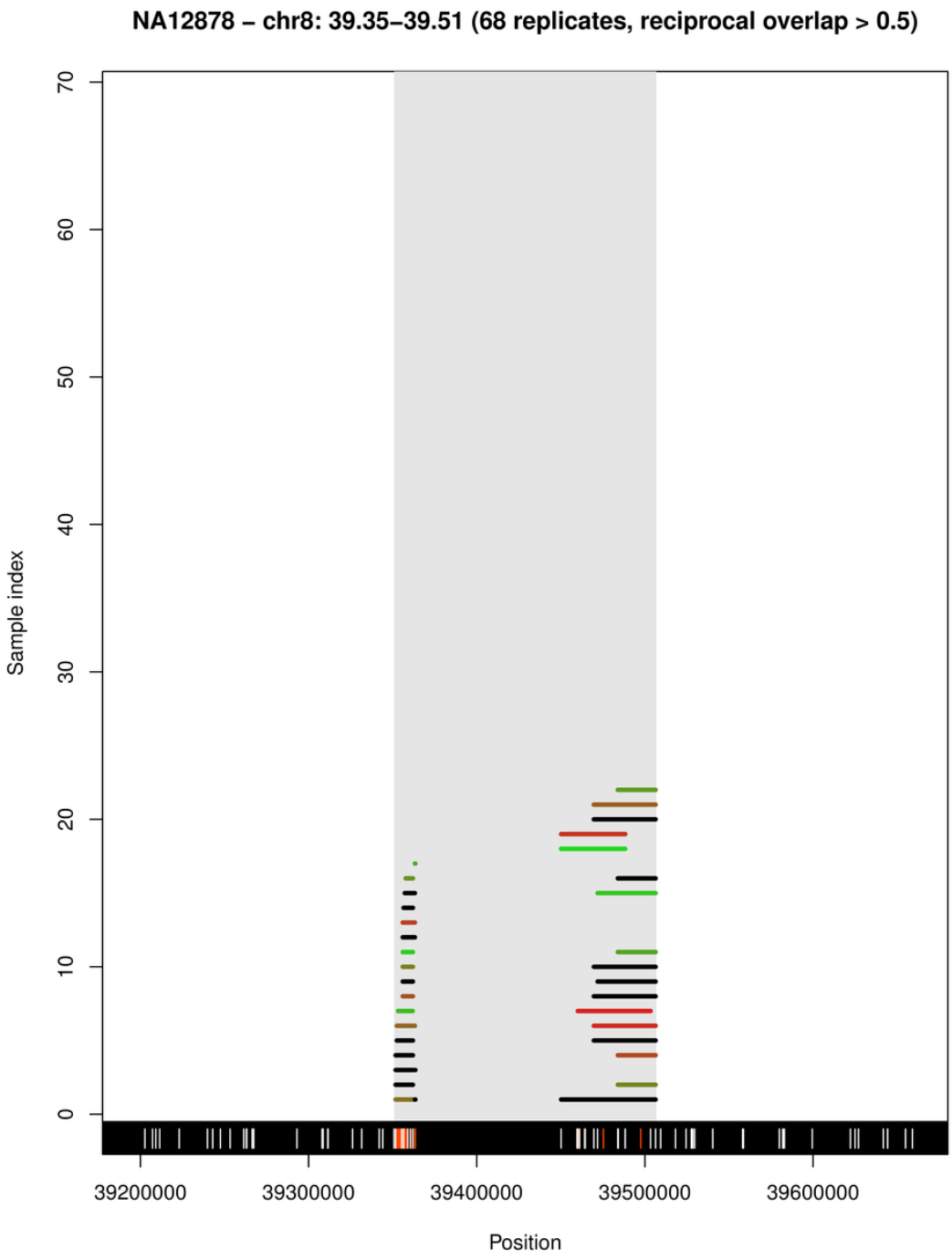


Figure S13: NA12878's raw CNV calls in known 1000 Genomes deletions (Illumina). Blue lines represent deletions that pass the confidence threshold (BF > 10) and the reciprocal overlap threshold (> 50%). Black lines are those that do not pass the reciprocal overlap threshold (<= 50%), but still over the confidence threshold. Dotted lines represent duplications. A range of color, from green to red, represent lower or higher values of BF, respectively, but still below the threshold. The black band at the bottom represents the different probes in that region, where white and orange ticks represent SNP and CN probes, respectively. The gray rectangle shows the exact region of the 1000 Genomes Project deletions.

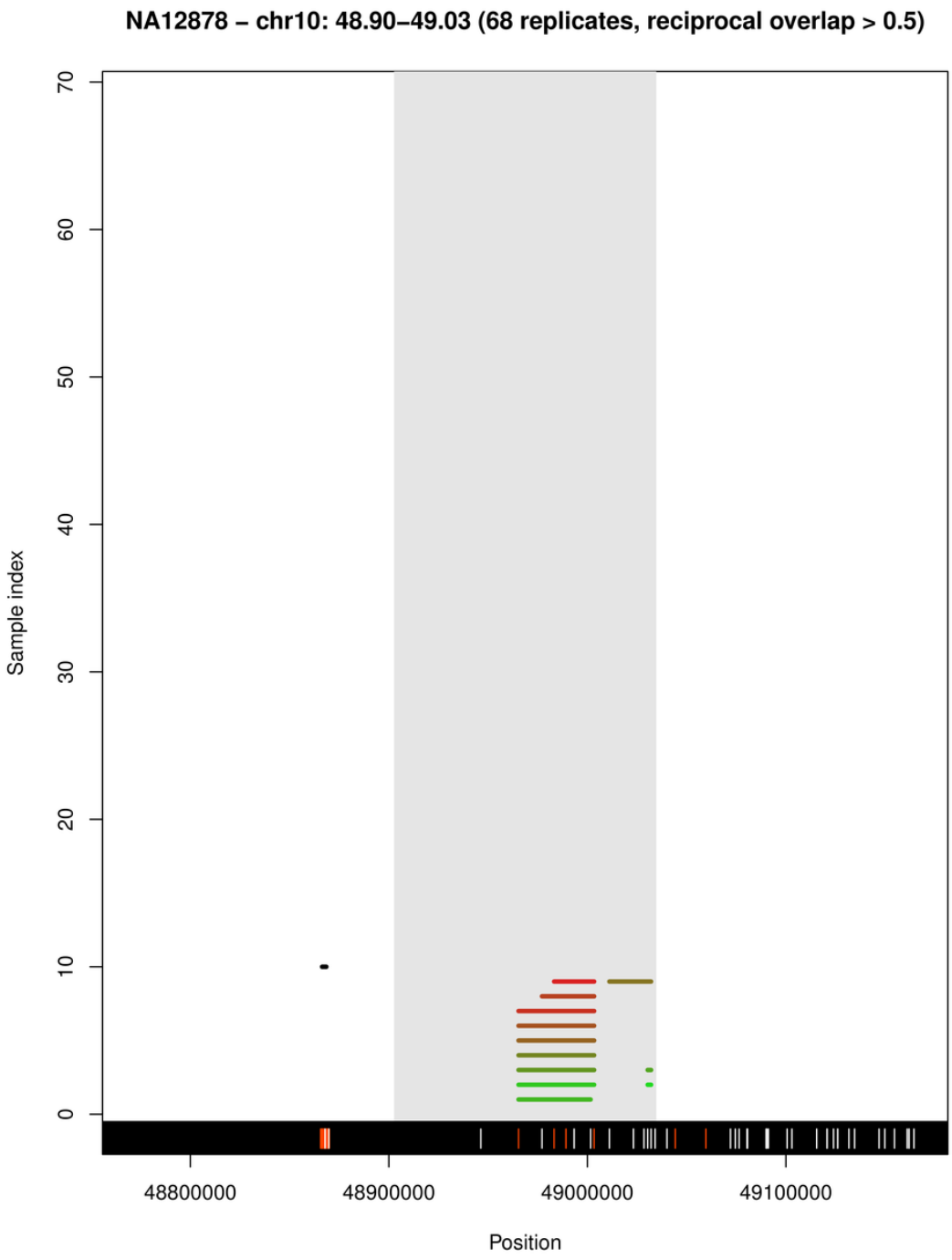


Figure S14: NA12878's raw CNV calls in known 1000 Genomes deletions (Illumina). Blue lines represent deletions that pass the confidence threshold (BF > 10) and the reciprocal overlap threshold (> 50%). Black lines are those that do not pass the reciprocal overlap threshold (<= 50%), but still over the confidence threshold. Dotted lines represent duplications. A range of color, from green to red, represent lower or higher values of BF, respectively, but still below the threshold. The black band at the bottom represents the different probes in that region, where white and orange ticks represent SNP and CN probes, respectively. The gray rectangle shows the exact region of the 1000 Genomes Project deletions.

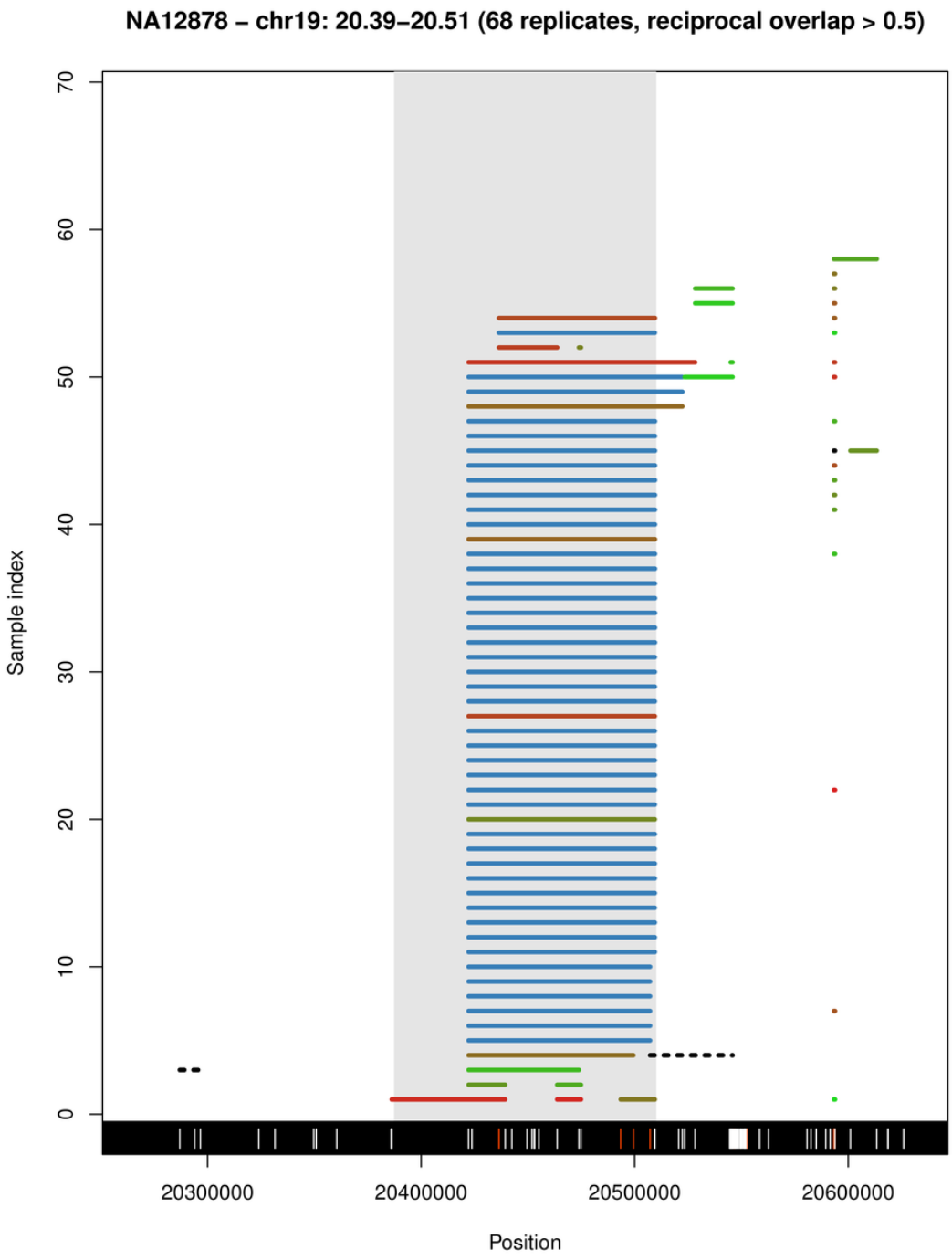
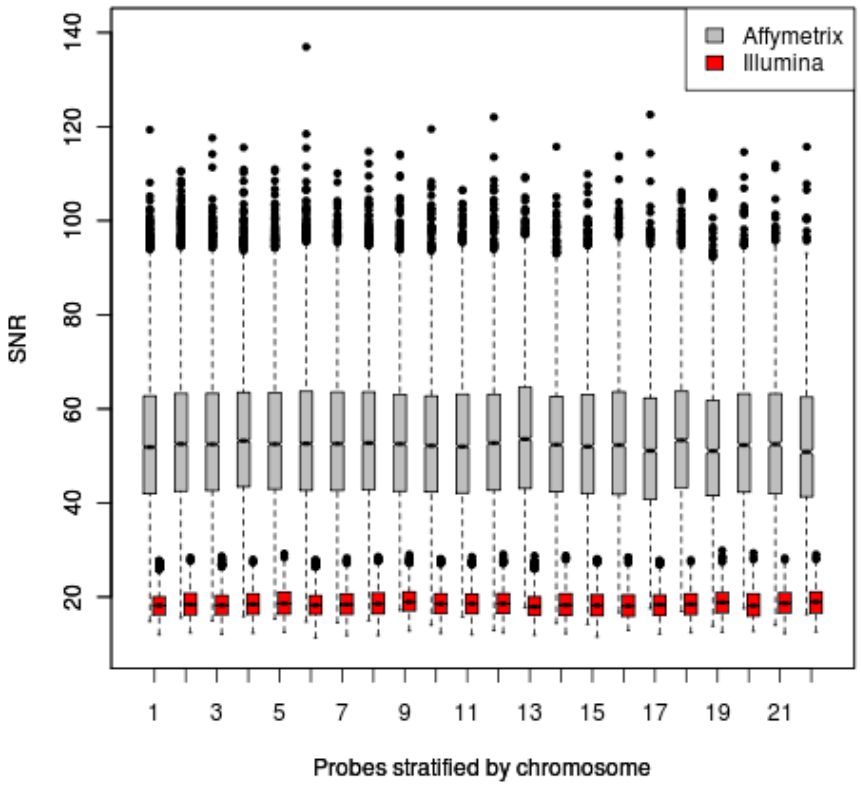
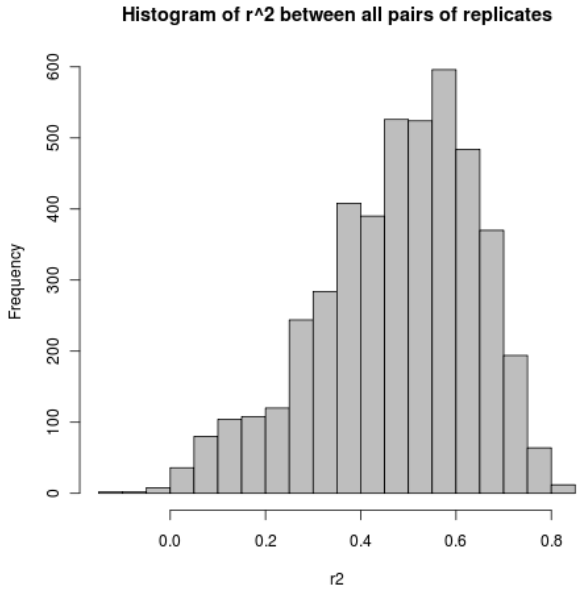


Figure S15: NA12878's raw CNV calls in known 1000 Genomes deletions (Illumina). Blue lines represent deletions that pass the confidence threshold (BF > 10) and the reciprocal overlap threshold (> 50%). Black lines are those that do not pass the reciprocal overlap threshold (<= 50%), but still over the confidence threshold. Dotted lines represent duplications. A range of color, from green to red, represent lower or higher values of BF, respectively, but still below the threshold. The black band at the bottom represents the different probes in that region, where white and orange ticks represent SNP and CN probes, respectively. The gray rectangle shows the exact region of the 1000 Genomes Project deletions.

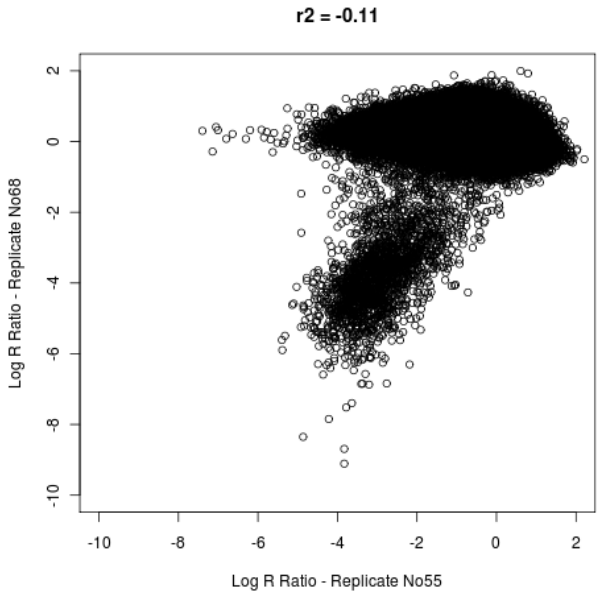


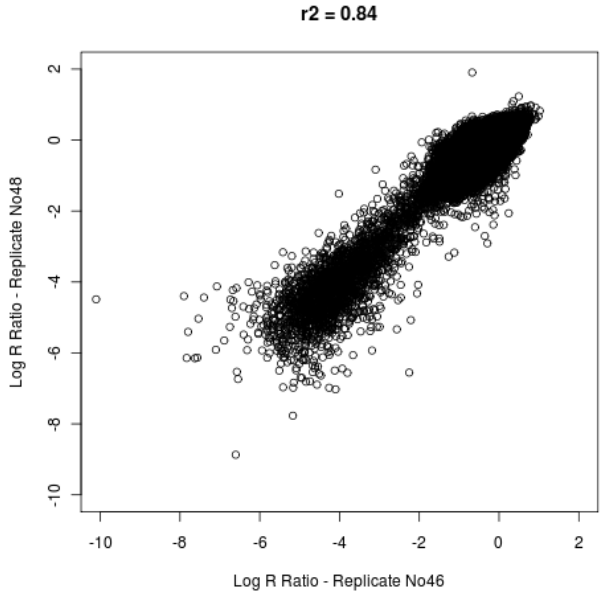
(a)

Figure S16: Signal-to-noise ratio comparison between platforms. An estimate of the Signal-to-Noise (SNR) is calculated for each probe as mean of total DNA intensity (log2(Xraw+Yraw)) over the standard deviation across the replicates of NA12878. Only the common probes between the two platforms were used (164,181 in total). All replicates typed in Illumina 660W were used (68) whereas 68 random replicates were selected from the total of 345 typed on Affymetrix 6.0.



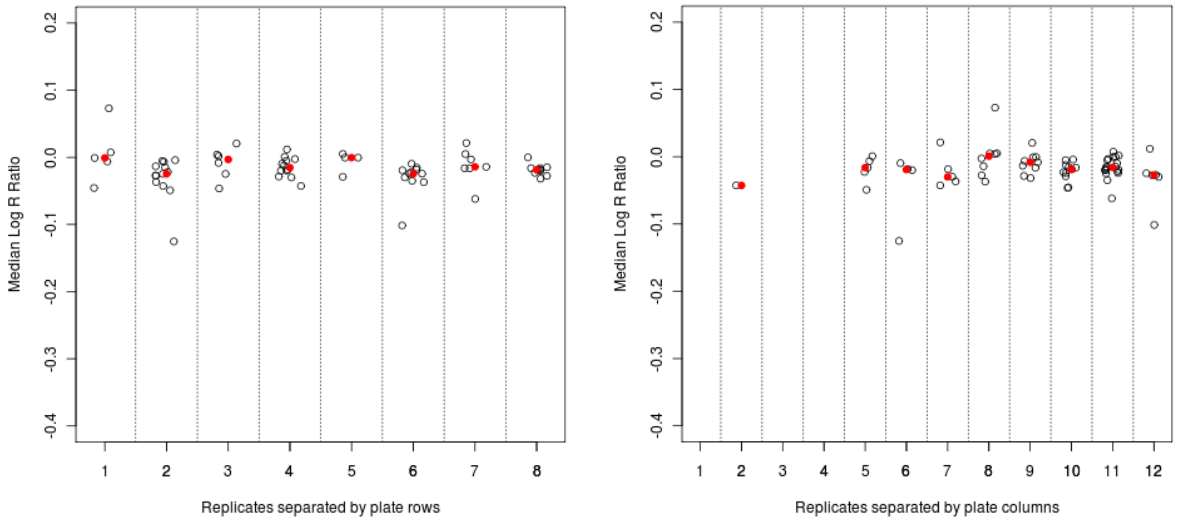
(a)





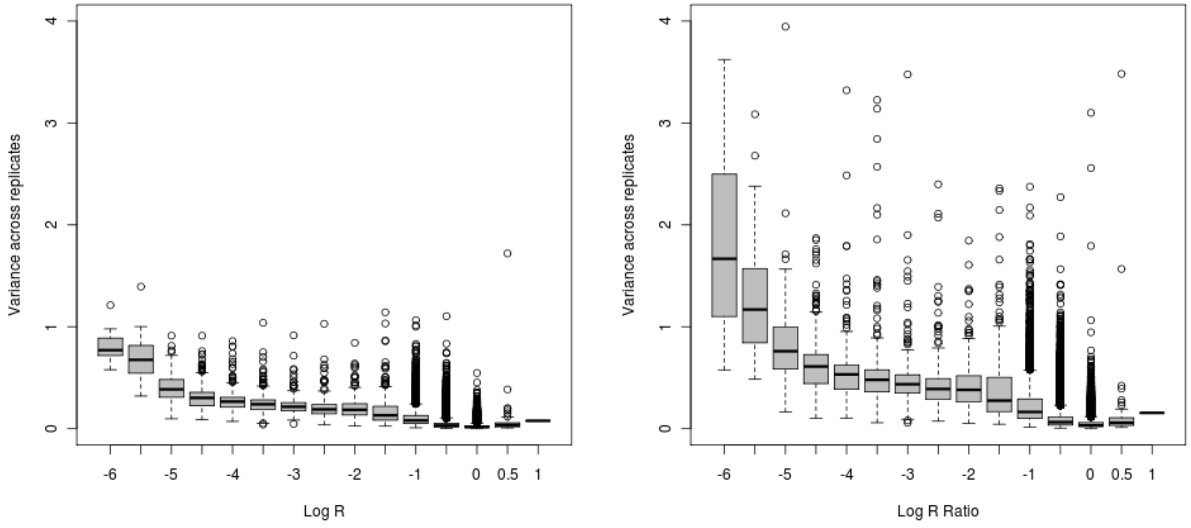
(b) **(c)**

Figure S17: Pearson correlation coefficients (r2) of standardised total DNA intensities (Log R Ratio) between all NA12878 replicates typed on Illumina 660W and scatter plot examples of two pairs of replicates. (a) Histogram of the pair-wise correlation coefficient between all replicates. (b) Scatter plot comparison between the total DNA intensity (Log R Ratio) of the replicate pair with the lowest correlation coefficient (r^2=-0.11). (c) Scatter plot comparison between the total DNA intensity (Log R Ratio) of the replicate pair with the highest correlation coefficient of all pairs of replicates (r^2=0.84).



(a) **(b)**

Figure S18: Plate row and column effect in total DNA intensity of NA12878 replicates typed on Illumina 660W. (a) The median Log R Ratio of each replicate is plotted against the row (jittered) of the plate the replicate was typed in. No row effect was evident (ANOVA p-value = 0.0849). The red points indicate the median Log R ratio of all replicates in each row. (b) The median Log R Ratio of each replicate is plotted against the column (jittered) of the plate the replicate was typed in. No column effect was evident (ANOVA p-value = 0.1184). Again, the red points indicate the median Log R ratio of all replicates in each column.



(a) **(b)**

Figure S19: Between-replicates variance comparison before and after standardisation. For each probe the variance across all replicates of (a) log R and (b) log R Ratio (standardised) is calculated. The probes are stratified by total DNA intensity. Due to variability between pool data used for standardisation, the variance of log R ratio is higher especially for the probes that measure background (low total DNA intensity).

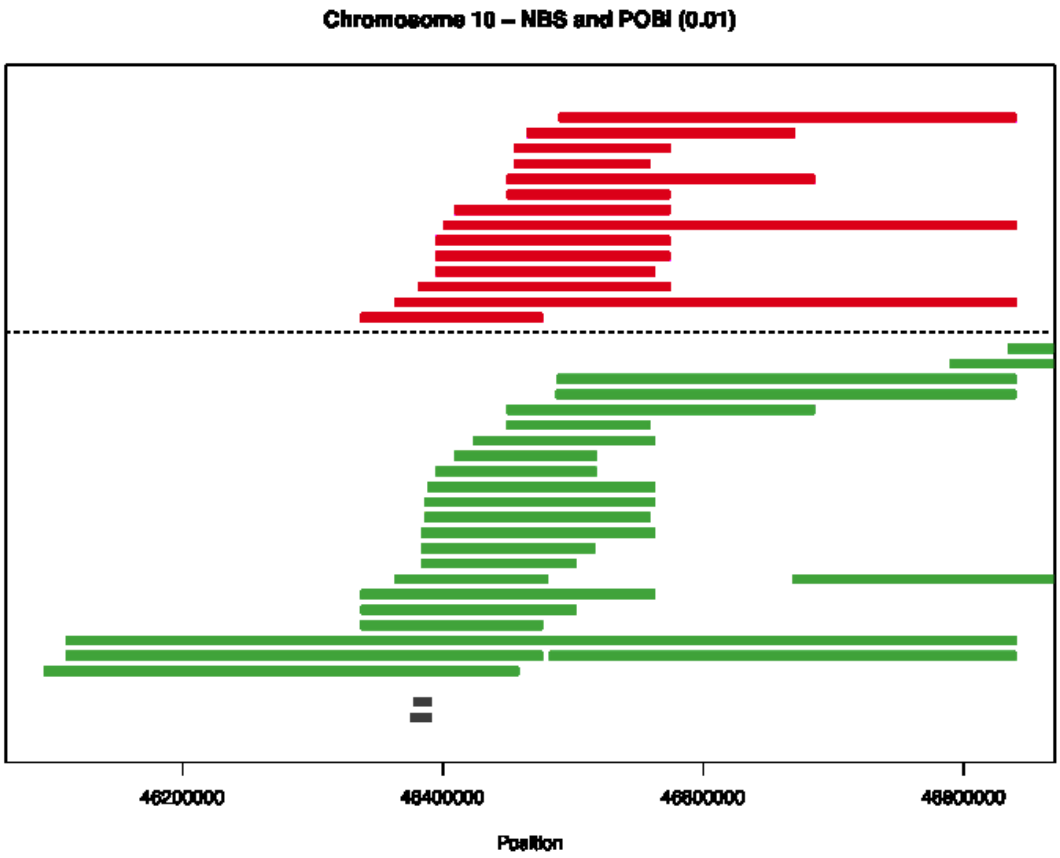


Figure S20: Visual representation of significant genes and the underlying $CNVE_{any}$ events. CNVs for the two groups are shown in green and red for POBI and NBS, respectively. The light colours indicate events that do not overlap with the gene. The black bars represent the CNVR created using the 50% reciprocal overlap definition. Gene transcripts are shown in gray bars at the bottom. (Affymetrix)

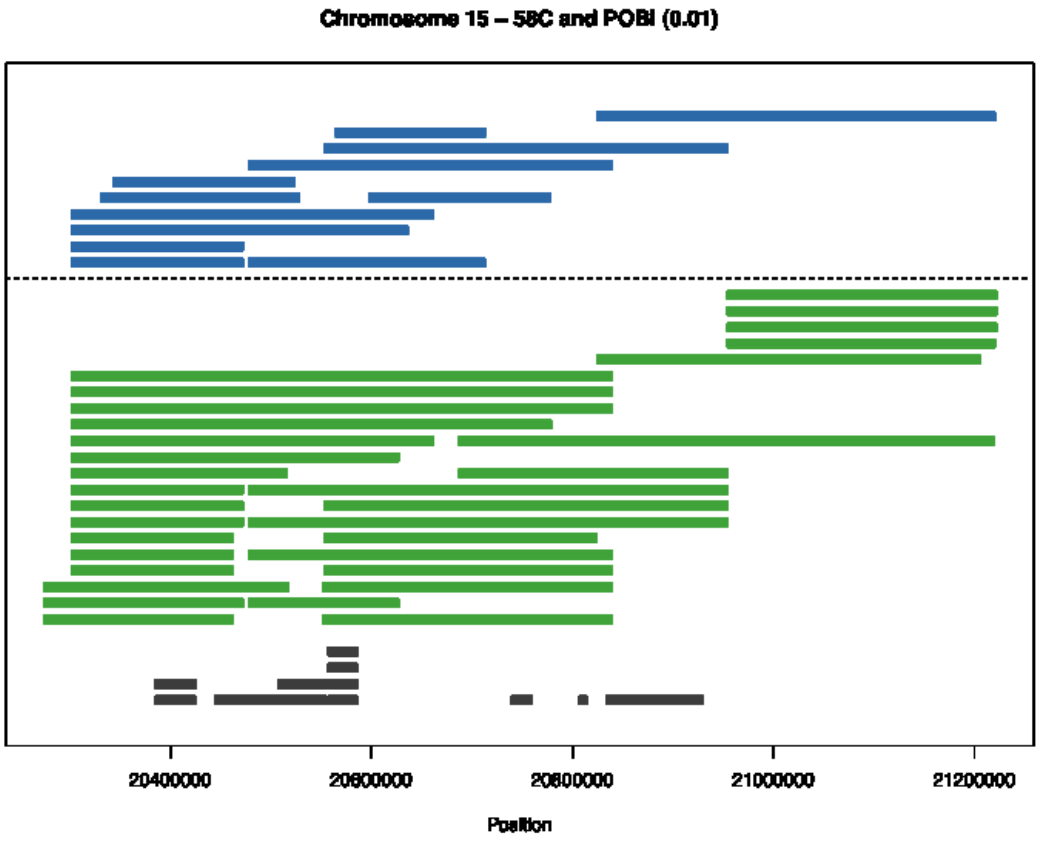


Figure S21: Visual representation of significant genes and the underlying $CNVE_{any}$ events. CNVs for the two groups are shown in green and blue for POBI and 58C, respectively. The light colours indicate events that do not overlap with the gene. The black bars represent the CNVR created using the 50% reciprocal overlap definition. Gene transcripts are shown in gray bars at the bottom. The second gene transcript is located between events that might have artificially split by the CNV detection algorithm(Illumina).

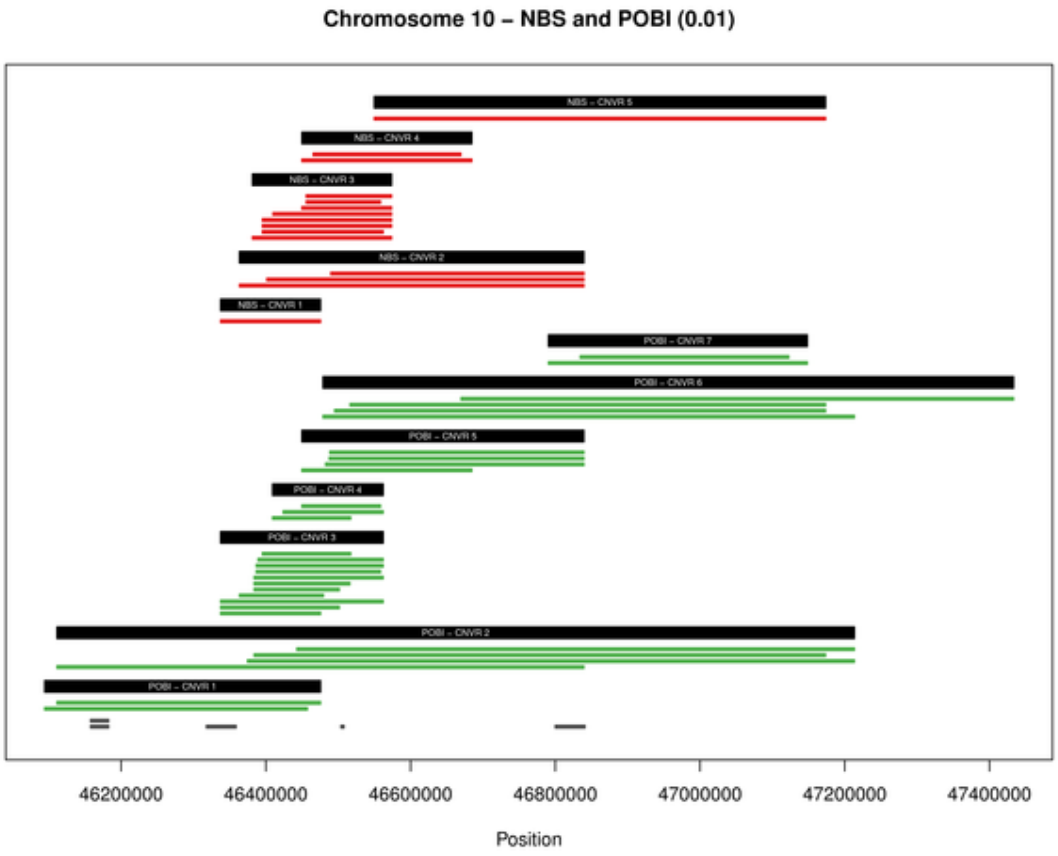


Figure S22: Visual representation of significant genes and the underlying $CNVR_{50}$ regions. CNVs for the two groups are shown in green and red for POBI and NBS, respectively. The black bars represent the CNVR created using the 50% reciprocal overlap definition. The first two genes (gray bars at the bottom) have a count of 6 CNVEs in POBI compared to 0 in NBS, giving a p value of 0.015. If the $CNVR_{any}$ definition was used, the counts would be 30 in POBI against 0 in NBS.

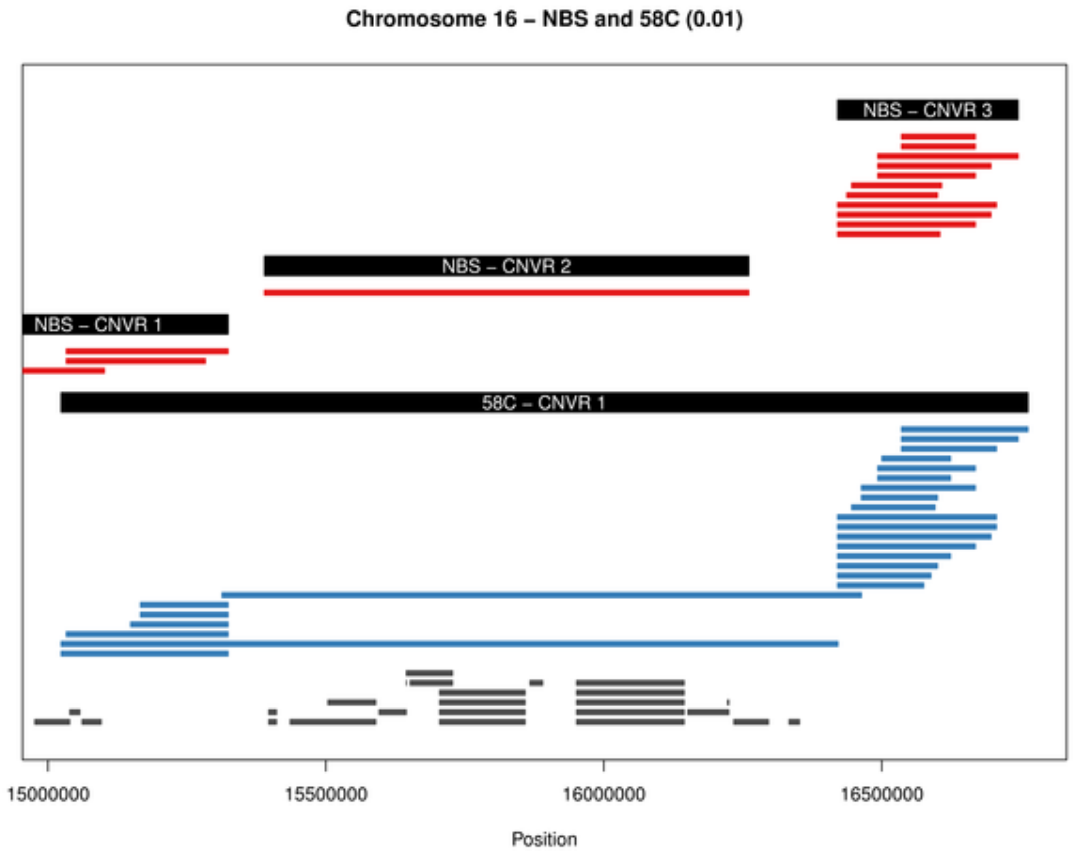


Figure S23: Visual representation of significant genes and the underlying $CNVR_{any}$ regions. CNVs for the two groups are shown in blue and red for 58C and NBS, respectively. The black bars represent the CNVR created using the any overlap definition. The CNVR in the 58C cohort, at the bottom of the figure, is larger than what it should be. We can clearly see two clusters of CNVEs, joined together by two larger CNVEs in the middle. The same chromosomal region in the NBS cohort shows 3 different CNVRs, because the CNVE in the middle is not long enough to join the two CNVE clusters, as in the 58C cohort.

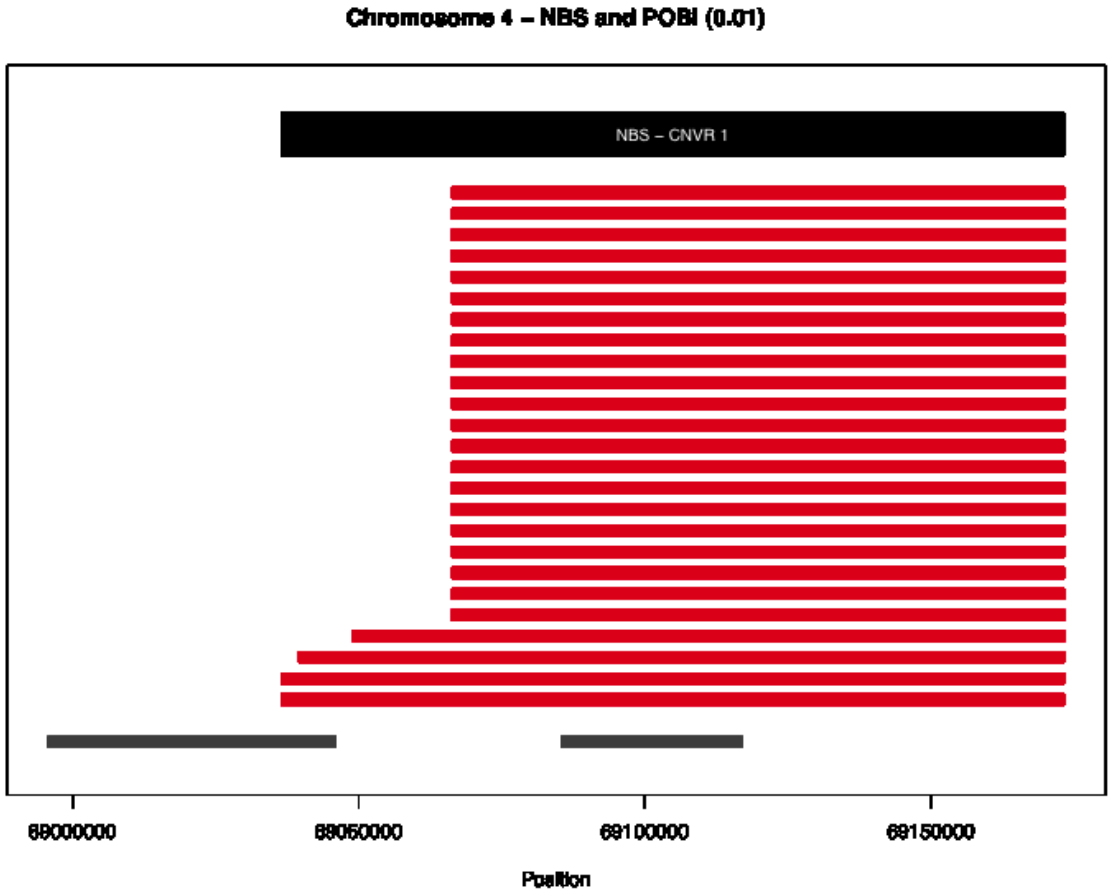


Figure S24: Visual representation of significant genes and the underlying $CNVR_{50}$ regions. CNVs are shown in red for NBS and no CNVs were detect in POBI. The black bars represent the CNVR created using the any 50% definition. Genes are indicated as gray bars at the bottom. In the case only the longest evnts in the region overlap the first gene yielding significant results.

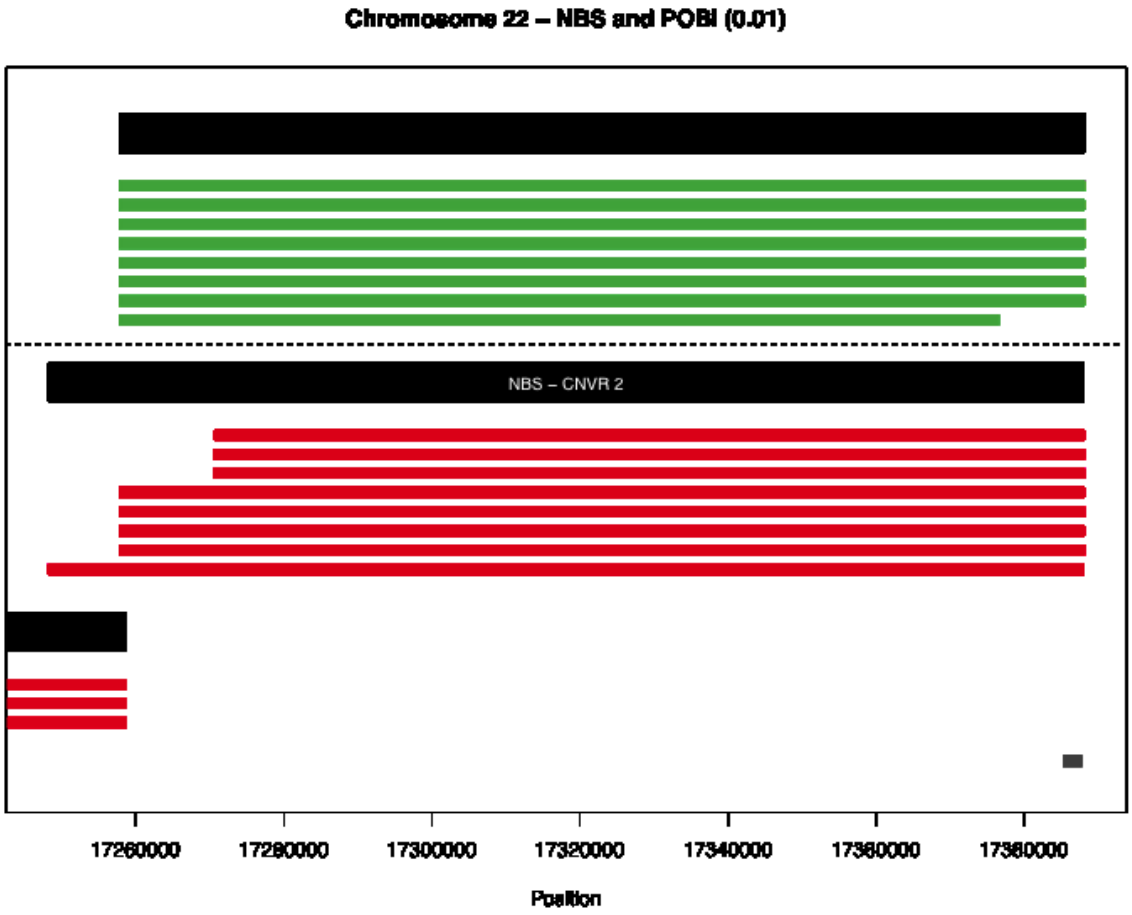


Figure S25: Visual representation of significant genes and the underlying $CNVR_{50}$ regions. CNVs are shown in red for NBS and green for POBI. The black bars represent the CNVR created using the any 50% definition with the extension of definin the breakpoints by the region covered by 90% of the events. None of the POBI CNVs were counted as overlapping since the region of 90% of the events is defined by the shorter CNV in the region. Genes are indicated as gray bars at the bottom. The gene was found erroneously as statistically significant.

Supplementary Tables

Table S-I: Deletions detected by 1000G (> 100 kb) in NA12878 used as "gold standard".

Chromosome	Start	End	GT
3	163987992	164111581	hetero
4	69063586	69191264	homo
4	70156653	70279881	hetero
5	12840861	13043434	hetero
8	39351231	39506383	hetero
10	48902961	49034384	hetero
19	20387650	20509815	hetero

Table S-II: Burden results (number of CNVs).

FOLD-CHANGE

Groups	Frequency	Affymetrix				Illumina			
		CNVEall	CNVE	CNVEany*	CNVE50§	CNVEall	CNVE	CNVEany*	CNVE50§
58C & POBI		1.071	1.007	0.995	1.015	0.750	1.034	1.112	0.845
NBS & 58C	0,05	0.967	1.019	1.047	1.006	1.052	1.259	1.156	1.188
NBS & POBI		1.035	1.025	1.028	1.020	0.790	1.314	1.286	1.077
58C & POBI		1.071	1.016	0.989	1.022	0.750	1.103	1.132	1.122
NBS & 58C	0,01	0.967	1.086	1.112	1.056	1.052	1.136	1.112	1.254
NBS & POBI		1.035	1.096	1.097	1.076	0.790	1.233	1.186	1.407

* Frequency of CNVEs calculated using the CNVRall definition

§ Frequency of CNVEs calculated using the CNVR50 definition

P-VALUES

Groups	Frequency	Affymetrix				Illumina			
		CNVEall	CNVE	CNVEany*	CNVE50§	CNVEall	CNVE	CNVEany*	CNVE50§
58C & POBI		2.424357e-17	0.768431	0.8221097	0.47072	6.102063e-55	0.1857417	9.568908e-05	9.213892e-14
NBS & 58C	0,05	1.742454e-05	0.403741	0.06164495	0.7867496	0.005198591	1.565904e-17	1.557787e-07	4.933273e-12
NBS & POBI		2.68594e-05	0.3012956	0.2670596	0.3569692	1.025374e-34	2.078445e-22	1.162342e-19	0.001589526
58C & POBI		2.424357e-17	0.6079369	0.7399676	0.44642	6.102063e-55	0.003872858	0.0009714431	0.0002034358
NBS & 58C	0,01	1.742454e-05	0.008729107	0.002050434	0.05761468	0.005198591	0.0002773788	0.004558087	1.768634e-12
NBS & POBI		2.68594e-05	0.005481388	0.009443198	0.01579043	1.025374e-34	8.205713e-10	3.289394e-06	9.626143e-26

EMPIRICAL P-VALUES

Groups	Frequency	Affymetrix				Illumina			
		CNVEall	CNVE	CNVEany*	CNVE50§	CNVEall	CNVE	CNVEany*	CNVE50§
58C & POBI		< 1e-06	0.771946	0.818004	0.471545	< 1e-06	0.187178	9e-05	< 1e-06
NBS & 58C	0,05	1.2e-05	0.405567	0.061326	0.785193	0.005093	< 1e-06	< 1e-06	< 1e-06
NBS & POBI		2.4e-05	0.302177	0.267097	0.355312	< 1e-06	< 1e-06	< 1e-06	0.001539
58C & POBI		< 1e-06	0.612679	0.73605	0.4512	< 1e-06	0.00362	0.000951	0.000184
NBS & 58C	0,01	1.2e-05	0.008417	0.002006	0.05619	0.005093	0.000232	0.004453	< 1e-06
NBS & POBI		2.4e-05	0.005285	0.009211	0.015579	< 1e-06	< 1e-06	4e-06	< 1e-06

Table S-III: Burden results (number of deletions).

FOLD-CHANGE

Groups	Frequency	Affymetrix				Illumina			
		CNVEall	CNVE	CNVEany*	CNVE50§	CNVEall	CNVE	CNVEany*	CNVE50§
58C & POBI	0,05	1.048	1.055	1.018	1.026	1.023	1.236	1.317	1.149
NBS & 58C		0.970	0.928	0.955	0.930	0.974	1.244	1.337	1.100
NBS & POBI		1.017	0.978	0.971	0.944	0.997	1.496	1.791	1.434
58C & POBI	0,01	1.048	1.041	1.029	1.066	1.023	1.339	1.357	1.370
NBS & 58C		0.970	0.980	0.989	0.962	0.974	1.155	1.103	1.222
NBS & POBI		1.017	1.018	1.025	1.014	0.997	1.421	1.251	1.671

* Frequency of CNVEs calculated using the CNVRall definition

§ Frequency of CNVEs calculated using the CNVR50 definition

P-VALUES

Groups	Frequency	Affymetrix				Illumina			
		CNVEall	CNVE	CNVEany*	CNVE50§	CNVEall	CNVE	CNVEany*	CNVE50§
58C & POBI	0,05	0.0001713552	0.1213228	0.6151089	0.3857693	0.2831274	5.414443e-10	2.736761e-13	8.032589e-06
NBS & 58C		0.01301763	0.02718107	0.1884507	0.01685528	0.2865884	8.579616e-10	3.395962e-15	0.002373887
NBS & POBI		0.1922553	0.5388034	0.4206333	0.07301164	0.8911	2.781277e-27	4.731556e-40	6.093099e-27
58C & POBI	0,01	0.0001713552	0.3376347	0.53683	0.1091082	0.2831274	2.472241e-10	4.123616e-10	1.063245e-13
NBS & 58C		0.01301763	0.6370703	0.8127772	0.3242861	0.2865884	0.000752474	0.0380813	2.712259e-07
NBS & POBI		0.1922553	0.6810077	0.5912241	0.7321752	0.8911	8.1196e-16	6.899521e-07	9.227908e-33

EMPIRICAL P-VALUES

Groups	Frequency	Affymetrix				Illumina			
		CNVEall	CNVE	CNVEany*	CNVE50§	CNVEall	CNVE	CNVEany*	CNVE50§
58C & POBI	0,05	0.000174	0.121212	0.61382	0.386038	0.283861	< 1e-06	< 1e-06	6e-06
NBS & 58C		0.013189	0.027508	0.189614	0.016436	0.285291	< 1e-06	< 1e-06	0.002205
NBS & POBI		0.190662	0.54495	0.412188	0.07132	0.890309	< 1e-06	< 1e-06	< 1e-06
58C & POBI	0,01	0.000174	0.340268	0.538325	0.107332	0.283861	< 1e-06	< 1e-06	< 1e-06
NBS & 58C		0.013189	0.645574	0.801309	0.326131	0.285291	0.00064	0.036684	< 1e-06
NBS & POBI		0.190662	0.684078	0.592968	0.741249	0.890309	< 1e-06	< 1e-06	< 1e-06

Table S-IV: Burden results (average CNV size).

FOLD-CHANGE

Groups	Frequency	Affymetrix				Illumina			
		CNVEall	CNVE	CNVEany*	CNVE50§	CNVEall	CNVE	CNVEany*	CNVE50§
58C & POBI	0,05	1,028	1,097	1,148	1,091	0,919	1,128	1,035	0,88
NBS & 58C		0,978	0,980	0,912	0,973	1,284	1,203	1,129	1,246
NBS & POBI		1,005	1,077	1,055	1,061	1,18	1,321	1,03	1,395
58C & POBI	0,01	1,028	1,072	1,141	1,115	0,919	1,124	1,098	1,077
NBS & 58C		0,978	1,020	0,923	0,949	1,284	1,031	0,96	1,263
NBS & POBI		1,005	1,083	1,023	1,059	1,18	1,11	1,019	1,316

* Frequency of CNVEs calculated using the CNVRall definition

§ Frequency of CNVEs calculated using the CNVR50 definition

P-VALUES

Groups	Frequency	Affymetrix				Illumina			
		CNVEall	CNVE	CNVEany*	CNVE50§	CNVEall	CNVE	CNVEany*	CNVE50§
58C & POBI	0,05	0.1109232	0.1173566	0.01091102	0.05918467	0.2802374	0.03522631	0.3575476	0.0001485202
NBS & 58C		0.1418114	0.5101386	0.7009328	0.8709662	6.325013e-20	4.507166e-08	6.646263e-06	3.320198e-11
NBS & POBI		0.8844405	0.2052592	0.07654976	0.1445594	1.814179e-18	1.482452e-12	0.2979181	1.042246e-21
58C & POBI	0,01	0.1109232	0.4936199	0.00733921	0.1062175	0.2802374	0.00701477	0.0822236	0.2249395
NBS & 58C		0.1418114	0.2198732	0.7847125	0.7216432	6.325013e-20	0.04507537	0.6314718	3.940675e-06
NBS & POBI		0.8844405	0.1730633	0.2975242	0.1350497	1.814179e-18	0.0003408082	0.1969879	2.960855e-07

EMPIRICAL P-VALUES

Groups	Frequency	Affymetrix				Illumina			
		CNVEall	CNVE	CNVEany*	CNVE50§	CNVEall	CNVE	CNVEany*	CNVE50§
58C & POBI	0,05	0.070176	0.096728	0.003432	0.039187	0.280025	0.028297	0.359553	1e-04
NBS & 58C		0.115532	0.527795	0.737562	0.882955	< 1e-06	< 1e-06	2e-06	< 1e-06
NBS & POBI		0.8866	0.185419	0.038591	0.103759	< 1e-06	< 1e-06	0.297714	< 1e-06
58C & POBI	0,01	0.070176	0.54367	0.004707	0.093685	0.280025	0.00563	0.079462	0.223148
NBS & 58C		0.115532	0.205483	0.843443	0.734671	< 1e-06	0.041341	0.632344	< 1e-06
NBS & POBI		0.8866	0.148273	0.269597	0.107663	< 1e-06	0.000173	0.194915	< 1e-06

Table S-V: Burden results (average deletion size).

FOLD-CHANGE

Groups	Frequency	Affymetrix				Illumina			
		CNVEall	CNVE	CNVEany*	CNVE50§	CNVEall	CNVE	CNVEany*	CNVE50§
58C & POBI	0,05	0,986	1,006	1,019	1,012	1,2	1,234	1,245	1,244
NBS & 58C		0,978	0,946	0,935	0,951	1,439	1,266	1,281	1,188
NBS & POBI		0,964	0,951	0,948	0,962	1,727	1,48	1,519	1,654
58C & POBI	0,01	0,986	0,921	0,916	1,012	1,2	1,234	1,264	1,246
NBS & 58C		0,978	1,009	1,015	0,920	1,439	1,122	0,965	1,132
NBS & POBI		0,964	0,926	0,924	0,929	1,727	1,278	1,198	1,335

* Frequency of CNVEs calculated using the CNVRall definition

§ Frequency of CNVEs calculated using the CNVR50 definition

P-VALUES

Groups	Frequency	Affymetrix				Illumina			
		CNVEall	CNVE	CNVEany*	CNVE50§	CNVEall	CNVE	CNVEany*	CNVE50§
58C & POBI	0,05	0.7086573	0.4119655	0.3553912	0.4297504	2.825223e-09	0.0005842959	0.003473923	2.555309e-06
NBS & 58C		0,4164436	0.4445593	0.4126132	0.4141433	3.806889e-32	2.64463e-07	1.509954e-06	1.069844e-05
NBS & POBI		0.05161602	0.680673	0.6161404	0.8286358	5.163596e-59	1.128123e-13	1.610102e-10	1.662317e-26
58C & POBI	0,01	0.7086573	0.9514936	0.7329489	0.4971016	2.825223e-09	0.001419219	0.001277773	0.0007796497
NBS & 58C		0,4164436	0.7252333	0.7141322	0.4189128	3.806889e-32	0.05561312	0.7656517	0.004642443
NBS & POBI		0.05161602	0.6637354	0.9765081	0.6217633	5.163596e-59	1.393128e-05	0.001586735	4.934804e-08

EMPIRICAL P-VALUES

Groups	Frequency	Affymetrix				Illumina			
		CNVEall	CNVE	CNVEany*	CNVE50§	CNVEall	CNVE	CNVEany*	CNVE50§
58C & POBI	0,05	0.936157	0.42316	0.223655	0.524379	< 1e-06	0.000322	0.002764	1e-06
NBS & 58C		0,477976	0.567882	0.384983	0.448087	< 1e-06	< 1e-06	< 1e-06	2e-06
NBS & POBI		0.033736	0.682312	0.618962	0.829232	< 1e-06	< 1e-06	< 1e-06	< 1e-06
58C & POBI	0,01	0.936157	0.952046	0.735603	0.701122	< 1e-06	0.000461	0.000421	0.000316
NBS & 58C		0,477976	0.72826	0.717288	0.469763	< 1e-06	0.050186	0.768772	0.003668
NBS & POBI		0.033736	0.665356	0.976803	0.622971	< 1e-06	2e-06	0.000846	< 1e-06

Table S-VI: Burden results (total length of CNVs).

FOLD-CHANGE

Groups	Frequency	Affymetrix				Illumina			
		CNVEall	CNVE	CNVEany*	CNVE50§	CNVEall	CNVE	CNVEany*	CNVE50§
58C & POBI	0,05	1,101	1,104	1,142	1,107	0,69	1,166	1,151	0,744
NBS & 58C		0,945	0,999	0,955	0,979	1,359	1,515	1,305	1,481
NBS & POBI		1,041	1,104	1,084	1,082	0,932	1,735	1,325	1,503
58C & POBI	0,01	1,101	1,089	1,128	1,14	0,69	1,24	1,243	1,208
NBS & 58C		0,945	1,108	1,026	1,003	1,359	1,171	1,068	1,583
NBS & POBI		1,041	1,187	1,122	1,139	0,932	1,369	1,208	1,851

* Frequency of CNVEs calculated using the CNVRall definition

§ Frequency of CNVEs calculated using the CNVR50 definition

P-VALUES

Groups	Frequency	Affymetrix				Illumina			
		CNVEall	CNVE	CNVEany*	CNVE50§	CNVEall	CNVE	CNVEany*	CNVE50§
58C & POBI	0,05	3.071878e-09	0.2348247	0.1592922	0.1217871	5.055152e-28	0.001764782	0.01275990	7.448422e-11
NBS & 58C		0.004505005	0.9865717	0.593285	0.7327879	1.559068e-16	2.099957e-18	2.934197e-08	7.952117e-15
NBS & POBI		0.01858276	0.1278727	0.1010597	0.1292652	0.03386556	1.441691e-27	1.242863e-10	7.682943e-16
58C & POBI	0,01	3.071878e-09	0.3257492	0.1805244	0.2151074	5.055152e-28	1.796353e-05	8.136036e-05	0.004810677
NBS & 58C		0.004505005	0.2617969	0.7599583	0.978591	1.559068e-16	0.001193851	0.1938441	1.824418e-11
NBS & POBI		0.01858276	0.06660247	0.1032279	0.1060198	0.03386556	8.591248e-11	0.0002234527	8.635827e-19

EMPIRICAL P-VALUES

Groups	Frequency	Affymetrix				Illumina			
		CNVEall	CNVE	CNVEany*	CNVE50§	CNVEall	CNVE	CNVEany*	CNVE50§
58C & POBI	0,05	< 1e-06	0.221922	0.102602	0.092628	< 1e-06	0.000854	0.00729	< 1e-06
NBS & 58C		0.001608	0.987886	0.650366	0.749513	< 1e-06	< 1e-06	< 1e-06	< 1e-06
NBS & POBI		0.0144	0.118541	0.091107	0.121818	0.033341	< 1e-06	< 1e-06	< 1e-06
58C & POBI	0,01	< 1e-06	0.328984	0.155023	0.196685	< 1e-06	2e-05	4.8e-05	0.002569
NBS & 58C		0.001608	0.258778	0.780905	0.980466	< 1e-06	0.000917	0.193236	< 1e-06
NBS & POBI		0.0144	0.053143	0.090124	0.094772	0.033341	< 1e-06	0.000138	< 1e-06

Table S-VII: Burden results (total length of deletions).

FOLD-CHANGE

Groups	Frequency	Affymetrix				Illumina			
		CNVEall	CNVE	CNVEany*	CNVE50§	CNVEall	CNVE	CNVEany*	CNVE50§
58C & POBI	0,05	1,033	1,061	1,038	1,039	1,228	1,525	1,639	1,429
NBS & 58C		0,949	0,877	0,893	0,884	1,402	1,575	1,712	1,307
NBS & POBI		0,980	0,931	0,921	0,904	1,721	2,215	2,721	2,371
58C & POBI	0,01	1,033	0,959	0,942	1,079	1,228	1,651	1,715	1,707
NBS & 58C		0,949	0,989	1,004	0,885	1,402	1,296	1,064	1,383
NBS & POBI		0,980	0,942	0,946	0,942	1,721	1,817	1,499	2,231

* Frequency of CNVEs calculated using the CNVRall definition

§ Frequency of CNVEs calculated using the CNVR50 definition

P-VALUES

Groups	Frequency	Affymetrix				Illumina			
		CNVEall	CNVE	CNVEany*	CNVE50§	CNVEall	CNVE	CNVEany*	CNVE50§
58C & POBI	0,05	0.1889201	0.5439179	0.7270113	0.5847038	5.885823e-08	2.569938e-12	1.923189e-12	5.048998e-12
NBS & 58C		0.2650028	0.1489135	0.2698414	0.05929071	1.155640e-13	2.539951e-13	8.399477e-16	1.081752e-07
NBS & POBI		0.0360929	0.3146387	0.2866601	0.08995026	6.93288e-34	4.020242e-31	8.750592e-33	9.475248e-48
58C & POBI	0,01	0.1889201	0.6418982	0.5610805	0.5072292	5.885823e-08	1.383364e-09	2.960599e-09	3.908398e-13
NBS & 58C		0.2650028	0.8633904	0.94773	0.2598456	1.155640e-13	0.0002824869	0.4105031	9.070185e-07
NBS & POBI		0.0360929	0.5081825	0.5760228	0.4661787	6.93288e-34	3.223610e-15	1.072977e-07	4.439005e-29

EMPIRICAL P-VALUES

Groups	Frequency	Affymetrix				Illumina			
		CNVEall	CNVE	CNVEany*	CNVE50§	CNVEall	CNVE	CNVEany*	CNVE50§
58C & POBI	0,05	0.181856	0.580374	0.758207	0.612077	< 1e-06	< 1e-06	< 1e-06	< 1e-06
NBS & 58C		0.263224	0.096109	0.234493	0.035846	< 1e-06	< 1e-06	< 1e-06	< 1e-06
NBS & POBI		0.024127	0.320515	0.289328	0.077623	< 1e-06	< 1e-06	< 1e-06	< 1e-06
58C & POBI	0,01	0.181856	0.663683	0.587468	0.542909	< 1e-06	< 1e-06	< 1e-06	< 1e-06
NBS & 58C		0.263224	0.863331	0.947737	0.234339	< 1e-06	0.000156	0.410882	< 1e-06
NBS & POBI		0.024127	0.531052	0.604643	0.4839	< 1e-06	< 1e-06	< 1e-06	< 1e-06

Table S-VIII: Burden results (Number of genes overlapping a CNV).

FOLD-CHANGE

Groups	Frequency	Affymetrix				Illumina			
		CNVEall	CNVE	CNVEany*	CNVE50§	CNVEall	CNVE	CNVEany*	CNVE50§
58C & POBI	0,05	1,148	1,012	1,020	1,004	1,085	1,147	1,166	1,193
NBS & 58C		0,909	1,059	1,073	1,042	1,171	1,403	1,225	1,363
NBS & POBI		1,044	1,071	1,080	1,048	1,271	1,744	1,375	1,645
58C & POBI	0,01	1,148	1,135	1,144	1,13	1,085	1,247	1,323	1,329
NBS & 58C		0,909	1,176	1,189	1,092	1,171	1,214	1,183	1,482
NBS & POBI		1,044	1,332	1,326	1,252	1,271	1,482	1,34	1,95

* Frequency of CNVEs calculated using the CNVRall definition

§ Frequency of CNVEs calculated using the CNVR50 definition

P-VALUES

Groups	Frequency	Affymetrix				Illumina			
		CNVEall	CNVE	CNVEany*	CNVE50§	CNVEall	CNVE	CNVEany*	CNVE50§
58C & POBI	0,05	6.320321e-15	0.8170362	0.7400526	0.9290926	0.008495997	0.01142299	0.01459212	0.0001742176
NBS & 58C		1.810712e-07	0.3326422	0.2921338	0.3919356	7.934064e-06	1.697498e-12	4.173242e-05	8.715848e-11
NBS & POBI		0.01318932	0.1637003	0.1503474	0.255908	1.245321e-12	4.71823e-24	1.532922e-09	1.48236e-24
58C & POBI	0,01	6.320321e-15	0.0961906	0.1185282	0.1563556	0.008495997	0.001276466	0.0001344668	0.0001131902
NBS & 58C		1.810712e-07	0.05185751	0.07195004	0.296173	7.934064e-06	0.003288192	0.01755917	8.793965e-09
NBS & POBI		0.01318932	0.0007594433	0.003441897	0.001861044	1.245321e-12	2.182222e-08	4.450999e-05	5.136145e-22

EMPIRICAL P-VALUES

Groups	Frequency	Affymetrix				Illumina			
		CNVEall	CNVE	CNVEany*	CNVE50§	CNVEall	CNVE	CNVEany*	CNVE50§
58C & POBI	0,05	< 1e-06	0.822976	0.749195	0.930343	0.006801	0.007518	0.008972	9.1e-05
NBS & 58C		< 1e-06	0.333939	0.291457	0.394612	3e-06	< 1e-06	4.1e-05	< 1e-06
NBS & POBI		0.01254	0.160455	0.145703	0.256063	< 1e-06	< 1e-06	< 1e-06	< 1e-06
58C & POBI	0,01	< 1e-06	0.090519	0.113206	0.141664	0.006801	0.000951	6.3e-05	3.4e-05
NBS & 58C		< 1e-06	0.042749	0.061273	0.295588	3e-06	0.002712	0.016266	< 1e-06
NBS & POBI		0.01254	0.000229	0.001378	0.000882	< 1e-06	< 1e-06	2.5e-05	< 1e-06

Table S-IX: Burden results (Number of genes overlapping a deletion).

FOLD-CHANGE

Groups	Frequency	Affymetrix				Illumina			
		CNVEall	CNVE	CNVEany*	CNVE50§	CNVEall	CNVE	CNVEany*	CNVE50§
58C & POBI	0,05	1,019	1,081	1,101	0,993	0,881	0,929	0,931	0,92
NBS & 58C		0,950	0,858	0,860	0,889	0,991	1,237	1,277	1,163
NBS & POBI		0,968	0,927	0,928	0,877	0,873	1,274	1,215	1,081
58C & POBI	0,01	1,019	1,097	1,192	1,14	0,881	1,154	1,218	1,179
NBS & 58C		0,950	0,981	0,997	0,929	0,991	1,233	1,138	1,338
NBS & POBI		0,968	1,078	1,167	1,056	0,873	1,399	1,29	1,667

* Frequency of CNVEs calculated using the CNVRall definition

§ Frequency of CNVEs calculated using the CNVR50 definition

P-VALUES

Groups	Frequency	Affymetrix				Illumina			
		CNVEall	CNVE	CNVEany*	CNVE50§	CNVEall	CNVE	CNVEany*	CNVE50§
58C & POBI	0,05	0.3537103	0.2944385	0.2325742	0.9026482	0.001875330	0.2714755	0.3417759	0.1600617
NBS & 58C		0.01263775	0.04629822	0.06515468	0.06966765	0.8465899	0.001773159	0.000556825	0.00881373
NBS & POBI		0.09197792	0.2540156	0.2942165	0.01799837	0.00112485	0.0002255638	0.01177854	0.1700856
58C & POBI	0,01	0.3537103	0.3162609	0.100353	0.1960038	0.001875330	0.2144036	0.1214479	0.08160197
NBS & 58C		0.01263775	0.836846	0.9789372	0.4726596	0.8465899	0.04394457	0.2731836	0.0003026386
NBS & POBI		0.09197792	0.3922827	0.1139513	0.5058582	0.00112485	0.000944574	0.02601906	3.584395e-10

EMPIRICAL P-VALUES

Groups	Frequency	Affymetrix				Illumina			
		CNVEall	CNVE	CNVEany*	CNVE50§	CNVEall	CNVE	CNVEany*	CNVE50§
58C & POBI	0,05	0.354531	0.295053	0.230264	0.90382	0.001689	0.271674	0.341397	0.15917
NBS & 58C		0.0117	0.040048	0.058096	0.066025	0.847631	0.001582	0.000395	0.008271
NBS & POBI		0.092015	0.252601	0.294477	0.017365	0.00094	0.000173	0.010041	0.16928
58C & POBI	0,01	0.354531	0.315278	0.095583	0.191784	0.001689	0.209224	0.108792	0.075771
NBS & 58C		0.0117	0.837126	0.97846	0.476491	0.847631	0.037342	0.273312	0.000145
NBS & POBI		0.092015	0.393102	0.111632	0.506682	0.00094	0.000417	0.019269	< 1e-06

Table S-X: Ig regions removed for QC.

Region Name	Chromosome	Start	End
IGK	2	87419380	89911535
TRG	7	38255369	38365141
TRB	7	141647264	142340942
TRA	14	21159851	22090936
IGH	14	105124272	106368585
IGL	22	20710462	21595082

Annexe III
Documentation de pyGenClean

pyGenClean Documentation

Release 1.2.3

Louis-Philippe Lemieux Perreault

March 14, 2013

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER

ONE

INTRODUCTION

Genetic association studies making use of high throughput genotyping arrays need to process large amounts of data in the order of millions of markers per experiment. The first step of any analysis with genotyping arrays is typically the conduct of a thorough data clean up and quality control to remove poor quality genotypes and generate metrics to inform and select individuals for downstream statistical analysis.

pyGenClean is an informatics tool to facilitate and standardize the genetic data clean up pipeline with genotyping array data. In conjuction with a source batch-queuing system, the tool minimizes data manipulation errors, it accelerates the completion of the data clean up process and it provides informative graphics and metrics to guide decision making for statistical analysis.

pyGenClean is a command tool working on both Linux and Windows operating systems. Its usage is shown below:

```
$ run_pyGenClean --help
usage: run_pyGenClean [-h] [--bfile FILE] [--tfile FILE] [--file FILE] --conf
                      FILE [--overwrite]

Runs the data clean up (version 1.2.3).

optional arguments:
  -h, --help    show this help message and exit

Input File:
  --bfile FILE  The input file prefix (will find the plink binary files by
                appending the prefix to the .bim, .bed and .fam files,
                respectively).
  --tfile FILE  The input file prefix (will find the plink transposed files by
                appending the prefix to the .tped and .tfam files,
                respectively).
  --file FILE   The input file prefix (will find the plink files by appending
                the prefix to the .ped and .fam files).

Configuration File:
  --conf FILE   The parameter file for the data clean up.

General Options:
  --overwrite   Overwrites output directories without asking the user.
                [DANGEROUS]
```

The tool consists of multiple standalone scripts that are linked together via a main script (`run_pyGenClean`) and a configuration file (the `--conf` option), the latter facilitating user customization.

The *Data clean up protocol schema* shows the proposed data cleanup pipeline. Each box represents a customizable standalone script with a quick description of its function. Optional manual checks for go-no-go decisions are indicated.

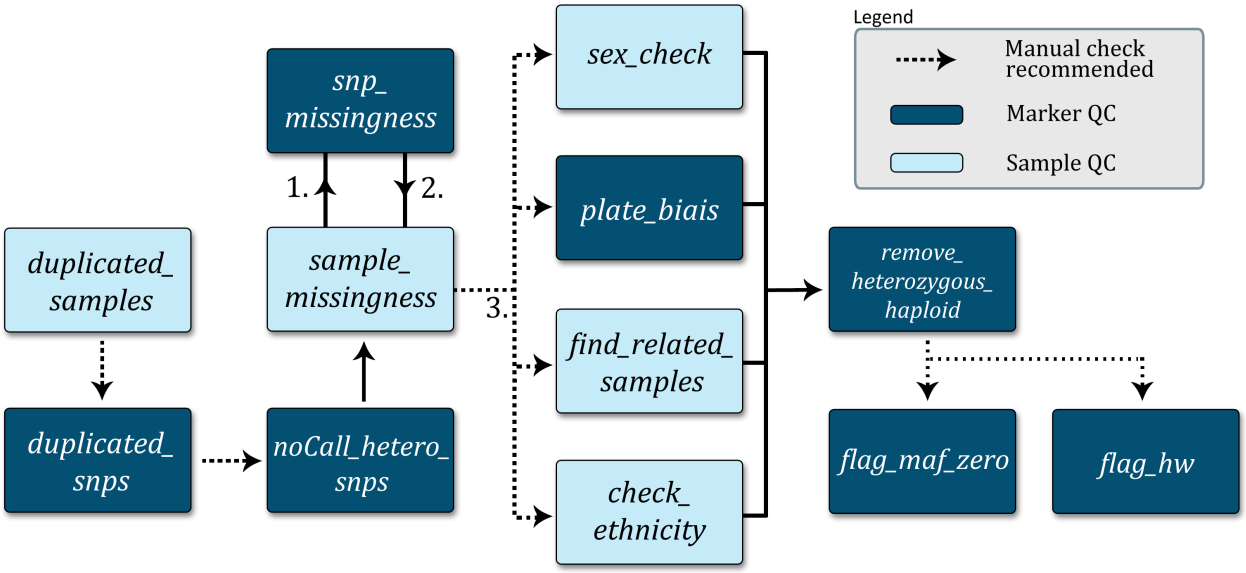


Figure 1.1: Data clean up protocol schema

CHAPTER

TWO

INSTALLATION

pyGenClean is a Python package that works on both Linux and Windows operating systems. It requires a set of Python dependencies and PLINK. Complete installation procedures are available for both Linux (32 and 64 bits) and Windows in the following sections.

2.1 Linux Installation

The following steps will help you install *pyGenClean* on a Linux machine. It has been fully tested on 64 bits machine, but should work without issue on a 32 bits machine.

2.1.1 Prerequisites

The following softwares and packages are required for *pyGenClean*:

1. Python 2.7
2. NumPy ($\geq$ 1.6.2)
3. matplotlib ($\geq$ 1.2.0)
4. scipy ($\geq$ 0.11.0)
5. scikit-learn ($\geq$ 0.12.1)
6. drmaa ($\geq$ 0.5)
7. PLINK (1.07)

2.1.2 Installation

Before installing any of those packages, you need to be sure that the development tools are installed on your machine. For example, for Fedora or CentOS installations:

```
$ sudo yum groupinstall "Development Tools"
$ sudo yum install zlib-devel
```

For a ubuntu installation:

```
$ sudo apt-get install build-essential libz-dev
```

Note: For all of the installation bellow, if the compilation completes without any error, everything should be fine. Note that on some platform (32 versus 64 bits), some test units might fail (scipy [double precision] and scikit-learn). Since not all of the functions provided by these modules are used, and that those that are used were tested, everything should run smoothly.

Python

Every Linux distributions comes with Python installed. However, this pre-installed Python can be obsolete. *pyGenClean* requires Python 2.7. To check the Python version, open a terminal and type the following command:

```
$ python -c "import platform; print(platform.python_version())"
```

Note: **If the version is 2.7.x**, be sure to install the development package. To do so, on Fedora or CentOS installation:

```
$ sudo yum install python-devel
```

And on ubuntu installation:

```
$ sudo apt-get install python-dev
```

You can then skip to the following section, the installation of a *Python Virtual Environment*.

If the version is lower than 2.7, or **if the version is 3.x**, you will need to install your own version of Python. To install your own version of Python, download the source code on the download page: http://python.org/download/ by selecting the bzipped source tarball. At the moment of writing this documentation, the latest version was 2.7.3.

Locate the downloaded archive file (usually in the `~/Downloads` directory, and performed the following command, in the terminal:

```
$ cd ~/Downloads
$ tar -jxf Python-2.7.3.tar.bz2
$ cd Python-2.7.3
```

Be sure that every conditions are met by reading the `README` file. Create a directory where Python will be install, and configure the installation accordingly:

```
$ mkdir -p ~/softwares/Python-2.7
$ ./configure --prefix=$HOME/softwares/Python-2.7
```

Once the configuration has been made, build Python:

```
$ make
```

After compilation, check if there are missing development packages. They will be enumerated after this line: `Python build finished, but the necessary bits to build these modules were not found`. Install the missing packages.

Once everything is build, check and install Python:

```
$ make test
$ make install
```

To test the new installation of Python, type the following command in the terminal:

```
$ ~/softwares/Python-2.7/bin/python
Python 2.7.3 (default, Feb 16 2013, 12:23:41)
[GCC 4.7.2 20121109 (Red Hat 4.7.2-8)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Type `exit()` to close the Python interpreter.

Python Virtual Environment

We recommend installing *pyGenClean* in a virtual environment. To do so, download the latest version of `virtualenv`, located at this web page: http://pypi.python.org/pypi/virtualenv. At the moment of writing this documentation, the latest version was 1.8.4, and the file was named `virtualenv-1.8.4.tar.gz`. Locate the archive, which is usually in the `~/Downloads` directory.

```
$ cd ~/Downloads
$ tar -zxf virtualenv-1.8.4.tar.gz
$ cd virtualenv-1.8.4
```

There is no need to install the module. Just create a directory to create the Python virtual environment:

```
$ mkdir -p ~/softwares/Python-2.7_virtualenv
```

If you have installed a new version on Python (that should be installed in `~/softwares/Python-2.7`, perform the following command to install the virtual environment:

```
$ ~/softwares/Python-2.7/bin/python ./virtualenv.py \
>     --no-site-packages \
>     ~/softwares/Python-2.7_virtualenv
```

Note: If you already had a Python 2.7 installation, perform the following command to install the virtual environment:

```
$ python ./virtualenv.py \
>     --no-site-packages \
>     ~/softwares/Python-2.7_virtualenv
```

Locate the `bin` directory of the Python virtual environment, and be certain that there is a file called `python2.7`.

```
$ cd ~/softwares/Python-2.7_virtualenv/bin
$ ls python2.7
```

If there are no `python2.7` file, create it using this command:

```
$ ln -s python python2.7
```

To activate the Python virtual environment, perform the following command:

```
$ source ~/softwares/Python-2.7_virtualenv/bin/activate
```

Finally, to deactivate the Python virtual environment, either close the terminal, or perform the following command:

```
$ deactivate
```

Warning: For the following installations and tests, be certain that the Python virtual environment is activated, or nothing will work as planned...
The best way to know if the Python virtual environment is activated, is to see its name, in parenthesis, before the usual prompt in the terminal. For example:

```
(Python-2.7_virtualenv)[username@localhost ~]$
```

NumPy

Before installing NumPy, be certain that the previously installed Python virtual environment is activated (see the *Python Virtual Environment* installation for more information).

Note: Be certain that `lapack`, `atlas` and `blas` development packages are installed. For Fedora or CentOS installation, perform the following command:

```
$ sudo yum install lapack-devel blas-devel atlas-devel
```

For ubuntu installation:

```
$ sudo apt-get install gfortran liblapack-dev libblas-dev libatlas-dev
```

Go to the NumPy download site (http://sourceforge.net/projects/numpy/files/) and download the latest version. At the moment of writing this documentation, the latest version was 1.7.0 and the file was named `numpy-1.7.0.tar.gz`. Do not install any beta with a `b` in the file name) or release candidate (with a `rc` in the file name) version.

Locate the downloaded archive (it should be in the `~/Downloads` directory). Perform the following commands:

```
$ cd ~/Downloads
$ tar -zxf numpy-1.7.0.tar.gz
$ cd numpy-1.7.0
```

Check NumPy dependencies by performing the following command:

```
$ python2.7 setup.py config
```

Build and install NumPy using these two commands:

```
$ python2.7 setup.py build
$ python2.7 setup.py install
```

If you want to test the NumPy installation, perform the following commands:

```
$ cd
$ pip install -U nose
$ python2.7 -c "import numpy; numpy.test()"
```

matplotlib

Before installing matplotlib, be certain that the previously installed Python virtual environment is activated (see *Python Virtual Environment* installation for more information).

Note: Be certain that `freetype` and `libpng` development packages are installed. For Fedora or CentOS installation, perform the following command:

```
$ sudo yum install freetype-devel libpng-devel
```

For ubuntu installation:

```
$ sudo apt-get install libfreetype6-dev libpng-dev
```

Go to the matplotlib download site (http://matplotlib.org/downloads.html) and download the latest version. At the moment of writing this documentation, the latest version was 1.2.0 and the file was named `matplotlib-1.2.0.tar.gz`. Do not install any beta (with a `b` in the file name) or release candidate (with a `rc` in the file name) version.

Locate the downloaded archive (it should be in the `~/Downloads` directory). Perform the following commands:

```
$ cd ~/Downloads
$ tar -zxf matplotlib-1.2.0.tar.gz
$ cd matplotlib-1.2.0
```

Check matplotlib dependencies by performing the following command:

```
$ python2.7 setup.py config
```

Build and install matplotlib using these two commands:

```
$ python2.7 setup.py build
$ python2.7 setup.py install
```

If you want to test the matplotlib installation, perform the following commands:

```
$ mkdir test_matplotlib
$ cd test_matplotlib
$ pip install -U nose
$ python2.7 -c "import matplotlib; matplotlib.test()"
```

scipy

Before installing scipy, be certain that the previously installed Python virtual environment is activated (see *Python Virtual Environment* installation for more information).

Go to the scipy download site (http://sourceforge.net/projects/scipy/files/) and download the latest version. At the moment of writing this documentation, the latest version was 0.11.0 and the file was named `scipy-0.11.0.tar.gz`. Do not install any beta (with a `b` in the file name) or release candidate (with a `rc` in the file name) version.

Locate the downloaded archive (it should be in the `~/Downloads` directory). Perform the following commands:

```
$ cd ~/Downloads
$ tar -zxf scipy-0.11.0.tar.gz
$ cd scipy-0.11.0
```

Check scipy dependencies by performing the following command:

```
$ python2.7 setup.py config
```

Build and install scipy using these two commands:

```
$ python2.7 setup.py build
$ python2.7 setup.py install
```

If you want to test the scipy installation, perform the following commands:

```
$ cd
$ pip install -U nose
$ python2.7 -c "import scipy; scipy.test()"
```

Warning: For reasons unknown (and out of our control), scipy failed a lot of unit tests on our installation of CentOS 6.3 (64 bits). If this is your case, and since scikit-learn uses scipy, you might want to double check the outliers detection in the *Ethnicity Module*. See below for more information (section *Testing the Algorithm*). We checked the results from the CentOS installation, and they were as expected.
If you're unsure of the outlier list, you can use another approach to find them using the MDS file that is generated by the *Ethnicity Module*.

scikit-learn

Before installing scikit-learn, be certain that the previously installed Python virtual environment is activated (see *Python Virtual Environment* installation for more information).

Go to the scikit-learn download site (http://sourceforge.net/projects/scikit-learn/files/) and download the latest version. At the moment of writing this documentation, the latest version was 0.13 and the file was named `scikit-learn-0.13.tar.gz`. Do not install any beta (with a `b` in the file name) or release candidate (with a `rc` in the file name) version.

Locate the downloaded archive (it should be in the `~/Downloads` directory). Perform the following commands:

```
$ cd ~/Downloads
$ tar -zxf scikit-learn-0.13.tar.gz
$ cd scikit-learn-0.13
```

Check scikit-learn dependencies by performing the following command:

```
$ python2.7 setup.py config
```

Build and install scikit-learn using these two commands:

```
$ python2.7 setup.py build
$ python2.7 setup.py install
```

If you want to test the scikit-learn installation, perform the following commands:

```
$ cd
$ pip install -U nose
$ python2.7 -c "import sklearn; sklearn.test()"
```

Don't worry if some tests fail. The required functions will be tested later with *pyGenClean*.

drmaa

This module is optional (only if you want to use *pyGenClean* on a server with a DRMAA-compliant distributed resource management system).

Before installing drmaa, be certain that the previously installed Python virtual environment is activated (see *Python Virtual Environment* installation for more information). To install the latest version of drmaa, perform the following command:

```
$ pip install drmaa
```

In order to use the drmaa module, a specific environment variable needs to be set. To check if the variable is set, simply execute the following command:

```
$ echo $DRMAA_LIBRARY_PATH
```

If nothing is displayed on the screen, you need to locate the file `libdrmaa.so` and set the variable to link to this path. To do so, add this line to the `~/.bash_profile` file (you will need to log out, so that the change takes effect):

```
export DRMAA_LIBRARY_PATH='/PATH/TO/THE/libdrmaa.so'
```

PLINK

If PLINK is not already install, go to the download page (http://pngu.mgh.harvard.edu/~purcell/plink/download.shtml), and download the 1.07 version for your Linux distribution (either x86_64 or i686). To find you distribution type, perform the following command:

```
$ uname -m
```

Locate the downloaded archive (it should be in the `~/Downloads` directory). Perform the following commands:

```
$ cd ~/Downloads
$ unzip plink-1.07*.zip
$ cd plink-1.07*/
$ mkdir -p ~/bin
$ cp plink ~/bin
```

Be sure that your newly created `bin` directory is in your `PATH`. To do so, perform the following command:

```
$ echo $PATH
```

If you see `/home/username/bin` in the path (where `username` is your user name), than you are good to go. If not, add the following two lines to you `~/.bash_profile` file (you will need to log out, so that the change takes effect):

```
PATH=$HOME/bin:$PATH
export PATH
```

To test the PLINK installation, perform the following commands and you should see the following results:

```
$ cd
$ plink --noweb
@----------------------------------------------------------@
|        PLINK!       |     v1.07      |   10/Aug/2009     |
|----------------------------------------------------------|
|  (C) 2009 Shaun Purcell, GNU General Public License, v2  |
|----------------------------------------------------------|
|  For documentation, citation & bug-report instructions:  |
|        http://pngu.mgh.harvard.edu/purcell/plink/        |
@----------------------------------------------------------@

Skipping web check... [ --noweb ]
Writing this text to log file [ plink.log ]
Analysis started: Sat Feb 16 14:20:09 2013

Options in effect:
    --noweb

Before frequency and genotyping pruning, there are 0 SNPs
```

```
0 founders and 0 non-founders found
0 SNPs failed missingness test ( GENO > 1 )
0 SNPs failed frequency test ( MAF < 0 )
After frequency and genotyping pruning, there are 0 SNPs

ERROR: Stopping as there are no SNPs left for analysis
```

pyGenClean

To install pyGenClean, download the latest version on the download site (http://www.statgen.org/main/index.php/Downloads/Downloads). At the moment of writing this documentation, the latest version was 1.2.3, and the name of the file is `pyGenClean-1.2.3.tar.gz`.

Locate the downloaded archive (it should be in the `~/Downloads` directory). Perform the following commands:

```
$ cd ~/Downloads
$ tar -zxf pyGenClean-1.2.3.tar.gz
$ cd pyGenClean-1.2.3
```

Build and install *pyGenClean* using these two commands:

```
$ python2.7 setup.py build
$ python2.7 setup.py install
```

2.1.3 Testing the Algorithm

Warning: Before using *pyGenClean*, be certain that the previously installed Python virtual environment is activated (see *Python Virtual Environment* installation for more information). If note, noting will work...

To test the algorithm, download the test data from http://www.statgen.org and the HapMap reference populations (build 37).

Locate the downloaded archives (it should be in the `~/Downloads` directory). Perform the following commands:

```
$ cd ~/Downloads
$ mkdir -p ~/test_pyGenClean
$ tar -C ~/test_pyGenClean -jxf check_ethnicity_HapMap_reference_populations_b37.tar.bz2
$ tar -C ~/test_pyGenClean -jxf pyGenClean_test_data.tar.bz2
$ cd ~/test_pyGenClean
```

Create a text file named `conf.txt` inside the `~/test_pyGenClean` directory, containing the following text:

```
[1]
script = check_ethnicity
ceu-bfile = check_ethnicity_HapMap_ref_pops_b37/hapmap_CEU_r23a_filtered_b37
yri-bfile = check_ethnicity_HapMap_ref_pops_b37/hapmap_YRI_r23a_filtered_b37
jpt-chb-bfile = check_ethnicity_HapMap_ref_pops_b37/hapmap_JPT_CHB_r23a_filtered_b37
nb-components = 2
multiplier = 1

[2]
script = sex_check
```

Run the following command:

```
$ run_pyGenClean \
>     --conf conf.txt \
>     --bfile pyGenClean_test_data/1000G_EUR-MXL_Human610-Quad-v1_H
```

Valuable information will be shown in the terminal. Once the program has finished, the results are in the new directory `data_clean_up.date_time` where `date` is the current date, and `time` is the time when the program started.

Here are the new directory structure, with only the files you might be interested in:

- `data_clean_up.data_time/`
  - `1_check_ethnicity/`
    - `ethnicity.before.png`
    - `ethnicity.outliers.png`
    - `ethnicity.outliers`
    - `ethnicity.population_file_outliers`
  - `2_sex_check/`
    - `sexcheck.list_problem_sex`

The first image in the first directory (*ethnicity.before.png*) shows the MDS values for each sample before outlier detection. The second image (*ethnicity.outliers.png*) shows the outliers that should be removed for further analysis. Finally, the file `ethnicity.outliers` include a list of samples that should be removed for further analysis. **The total number of outliers for this test should be exactly 62**, but the figures might be mirrored for 32 bits systems. For more information about the results of this module, refer to Section *Ethnicity Module*.

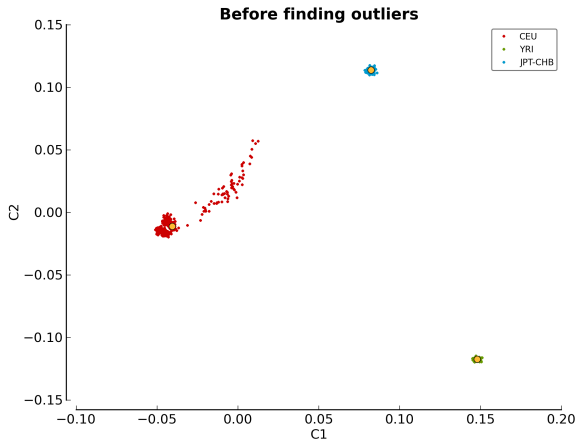


Figure 2.1: ethnicity.before.png

In the second directory, there should be a file containing the list of samples with gender problem. **There should be exactly 4 samples with gender problem.** For more information about this module, refer to Section *Sex Check Module*.

If you want to compare your results with the expected ones, just download the files in the archive `pyGenClean_expected_results.tar.bz2`, available through http://www.statgen.org. They were generated using Fedora 18 (64 bits) in about 20 minutes. You should at least compare the following files:

- `1_check_ethnicity`
  - `ethnicity.outliers`

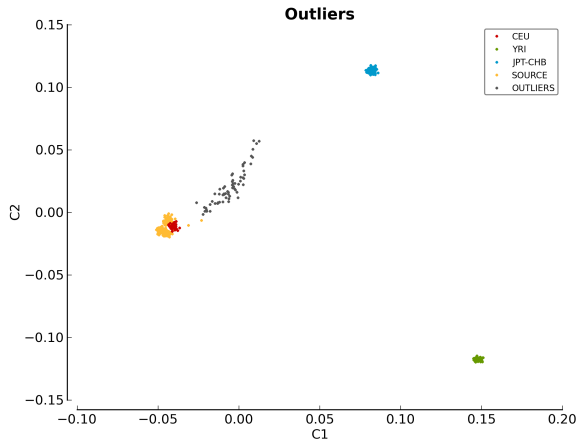


Figure 2.2: ethnicity.outliers.png

  - `ethnicity.population_file_outliers`
  - All the figures (they might be mirrored).
- `2_sex_check`
  - `sexcheck.list_problem_sex`
  - `sexcheck.list_problem_sex_ids`

2.2 Windows Installation

The following steps will help you install *pyGenClean* on a Windows machine. It has been tested on both Windows XP and Windows 7.

2.2.1 Prerequisites

The following softwares and packages are required for *pyGenClean*:

1. Python 2.7 (32 bits)
2. NumPy ($\geq$ 1.6.2)
3. matplotlib ($\geq$ 1.2.0)
4. scipy ($\geq$ 0.11.0).
5. scikit-learn ($\geq$ 0.12.1)
6. PLINK (1.07)

2.2.2 Installation

Python

On the Python download page (available at http://python.org/download/), download the latest version of Python 2.7 Windows Installer (**do not choose** the Windows X86-64 or any Python 3 installer). At the moment of writing this

documentation, the latest version was 2.7.3.

Locate the downloaded installer, and install it for all users using the default options. The directory `C:\Python27` should now be created.

Edit the `Path` variable to include the `C:\Python27` path. To do so, right click on `My Computer > Properties` on the Desktop (Windows XP users), or right click on `Computer > Properties` in the start menu and on `Advanced system parameters` (Windows 7 users). In the `Advanced` tab, click on the `Environment variables` button (see figure *System Properties*).

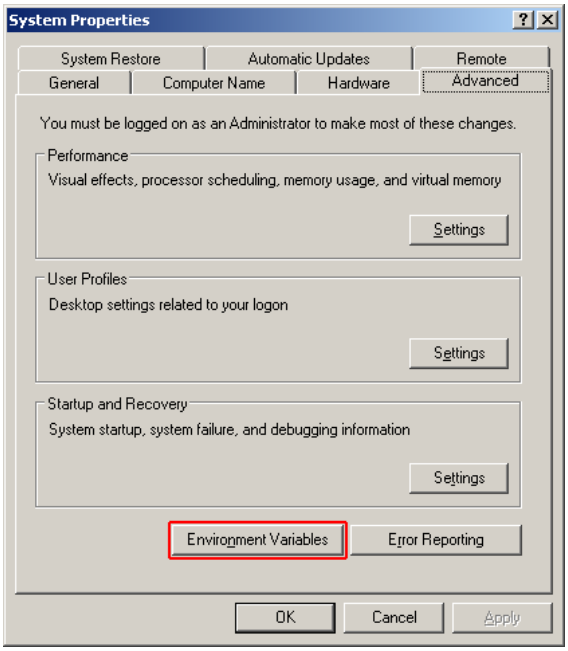


Figure 2.3: System Properties

Edit the `Path` variable (by selecting it and clicking on `Edit`, and add the following text at the end of the line: `;C:\Python27` (**do not forget** the `;`) (see figure *Edit System Variable*).

To test the installation, click on `Run...` in the start menu and type `cmd` (Windows XP users), or search for `cmd` in the search bar of the start menu (Windows 7 users). In the prompt, type `python`, and you should get the following result:

```
> python
Python 2.7.3 (default, Apr 10 2012, 23:31:26) [MSC v.1500 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Type `exit()` to close Python, and `exit` to close the command prompt.

NumPy

On the NumPy download page (available at http://sourceforge.net/projects/numpy/files/), download the latest version of Numpy (which is **not** a release candidate) by selecting the correct build (NumPy for Python 2.7 on win32). At the moment of writing this documentation, the latest version was 1.7.0, and the file was named `numpy-1.7.0-win32-superpack-python2.7.exe`. Do not install any beta (with a `b` in the file name) or release candidate (with a `rc` in the file name) version.

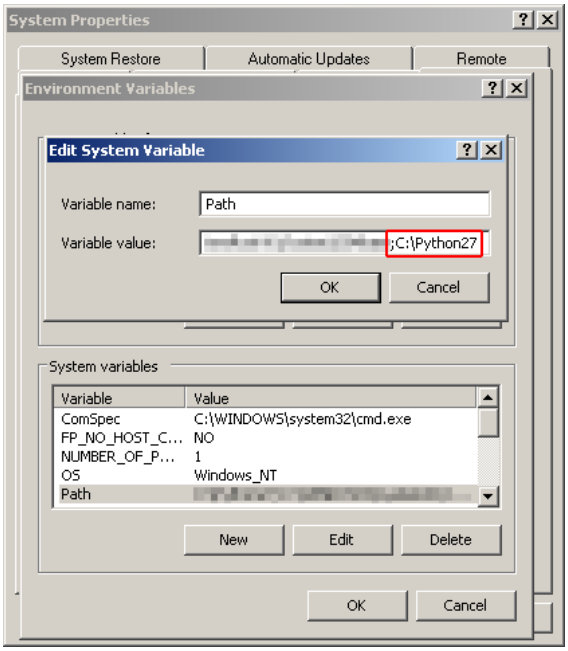


Figure 2.4: Edit System Variable

Locate the downloaded installer, and install it for all users using the default options. It should locate and install in the `C:\Python27` directory.

matplotlib

On the matplotlib download page (available at http://matplotlib.org/downloads.html), download the latest version of matplotlib by selecting the correct build (matplotlib for Python 2.7 on win32). At the moment of writing this documentation, the latest version was 1.2.0, and the file was named `matplotlib-1.2.0.win32-py2.7.exe`. Do not install any beta (with a `b` in the file name) or release candidate (with a `rc` in the file name) version.

Locate the downloaded installer, and install it for all users using the default options. It should locate and install in the `C:\Python27` directory.

scipy

On the scipy download page (available at http://sourceforge.net/projects/scipy/files/), download the latest version of scipy (which is **not** a release candidate) by selecting the correct build (scipy for Python 2.7 on win32). At the moment of writing this documentation, the latest version was 0.11.0, and the file was named `scipy-0.11.0-win32-superpack-python2.7.exe`. Do not install any beta (with a `b` in the file name) or release candidate (with a `rc` in the file name) version.

Locate the downloaded installer, and install it for all users using the default options. It should locate and install in the `C:\Python27` directory.

scikit-learn

On the scikit-learn download page (available at http://sourceforge.net/projects/scikit-learn/files/), download the latest version of scikit-learn by selecting the correct build (scikit-learn for Python 2.7 on win32). At the moment of writing this documentation, the latest version was 0.13, and the file was named

`scikit-learn-0.13.win32-py2.7.exe`. Do not install any beta (with a `b` in the file name) or release candidate (with a `rc` in the file name) version.

Locate the downloaded installer, and install it for all users using the default options. It should locate and install in the `C:\Python27` directory.

PLINK

On the PLINK download page (available at http://pngu.mgh.harvard.edu/~purcell/plink/download.shtml), download the latest version of PLINK for MS-DOS. At the moment of writing this documentation, the latest version was 1.07.

Locate the downloaded archive, and extract it directly in the `C:` directory. The name of the directory should be `C:\plink-1.07-dos`.

Edit the `Path` variable to include the `C:\plink-1.07-dos` path. Edit the `Path` variable by adding the following text at the end of the line: `;C:\plink-1.07-dos` (**do not forget** the `;`). For more information, refer to the *Python* installation section.

To test the installation, click on `Run...` in the start menu and type `cmd` (Windows XP users), or search for `cmd` in the search bar of the start menu (Windows 7 users). In the prompt, type `plink`, and you should get the following result:

```
> plink
@----------------------------------------------------------@
|        PLINK!       |     v1.07      |   10/Aug/2009     |
|-----------------------------------------------------------|
|  (C) 2009 Shaun Purcell, GNU General Public License, v2  |
|-----------------------------------------------------------|
|  For documentation, citation & bug-report instructions:  |
|        http://pngu.mgh.harvard.edu/purcell/plink/        |
@----------------------------------------------------------@

Web-based version check ( --noweb to skip )
Connecting to web...  OK, v1.07 is current

Writing this text to log file [ plink.log ]
Analysis started: Fri Feb 15 13:34:55 2013

Options in effect:

Before frequency and genotyping pruning, there are 0 SNPs
0 founders and 0 non-founders found
0 SNPs failed missingness test ( GENO > 1 )
0 SNPs failed frequency test ( MAF < 0 )
After frequency and genotyping pruning, there are 0 SNPs

ERROR: Stopping as there are no SNPs left for analysis
```

Type `exit` to close the command prompt.

pyGenClean

Just download the Windows installer, and install the software.

2.2.3 Testing the Algorithm

To test the algorithm, download the test data from http://www.statgen.org and the HapMap reference populations (build 37). Create a directory on your Desktop named `pyGenClean_test`, and extract the two archive into it. You should have the following directory structure:

```
Desktop\
    pyGenClean_test_data\
        1000G_EUR-MXL_Human610-Quad-v1_H.bed
        1000G_EUR-MXL_Human610-Quad-v1_H.bim
        1000G_EUR-MXL_Human610-Quad-v1_H.fam
    check_ethnicity_HapMap_ref_pops_b37\
        hapmap_CEU_r23a_filtered_b37.bed
        hapmap_CEU_r23a_filtered_b37.bim
        hapmap_CEU_r23a_filtered_b37.fam
        hapmap_YRI_r23a_filtered_b37.bed
        hapmap_YRI_r23a_filtered_b37.bim
        hapmap_YRI_r23a_filtered_b37.fam
        hapmap_JPT_CHB_r23a_filtered_b37.bed
        hapmap_JPT_CHB_r23a_filtered_b37.bim
        hapmap_JPT_CHB_r23a_filtered_b37.fam
```

Open the command prompt (see section *Python* for more information), and navigate to the newly created directory, and created an new text file using `notepad`:

```
> cd Desktop\pyGenClean_test
> notepad conf.txt
```

Insert the following code in the file:

```
[1]
script = check_ethnicity
ceu-bfile = check_ethnicity_HapMap_ref_pops_b37/hapmap_CEU_r23a_filtered_b37
yri-bfile = check_ethnicity_HapMap_ref_pops_b37/hapmap_YRI_r23a_filtered_b37
jpt-chb-bfile = check_ethnicity_HapMap_ref_pops_b37/hapmap_JPT_CHB_r23a_filtered_b37
nb-components = 2
multiplier = 1

[2]
script = sex_check
```

Finally, run the following command:

```
> python C:\Python27\Scripts\run_pyGenClean ^
      --conf conf.txt ^
      --bfile pyGenClean_test_data\1000G_EUR-MXL_Human610-Quad-v1_H
```

Valuable information will be shown on the command prompt. Once the program has finished, the results are in the new directory `data_clean_up.date_time` where `date` is the current date, and `time` is the time when the program started.

Here are the new directory structure, with only the files you might be interested in:

- `data_clean_up.data_time\`
  - `1_check_ethnicity\`
    - `ethnicity.before.png`
    - `ethnicity.outliers.png`
    - `ethnicity.outliers`

    * ethnicity.population_file_outliers

- 2_sex_check\

    * sexcheck.list_problem_sex

The first image in the first directory (*ethnicity.before.png*) shows the MDS values for each sample before outlier detection. The second image (*ethnicity.outliers.png*) shows the outliers that should be removed for further analysis. Finally, the file `ethnicity.outliers` include a list of samples that should be removed for further analysis. **The total number of outliers for this test should be exactly 62**. For more information about the results of this module, refer to Section *Ethnicity Module*.

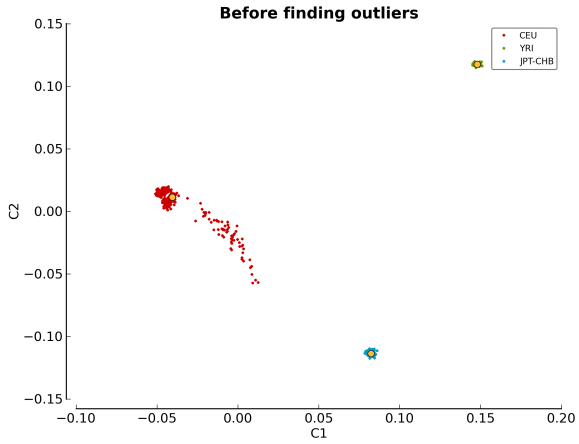


Figure 2.5: ethnicity.before.png

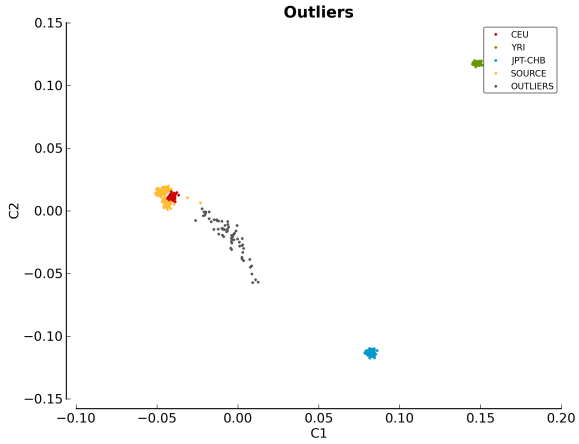


Figure 2.6: ethnicity.outliers.png

In the second directory, there should be a file containing the list of samples with gender problem. **There should be exactly 4 samples with gender problem.** For more information about this module, refer to Section *Sex Check Module*.

If you want to compare your results with the expected ones, just download the files in the archive `pyGenClean_expected_results.tar.bz2`, available through http://www.statgen.org. They were generated using Fedora 18 (64 bits) in about 20 minutes. You should at least compare the following files:

- 1_check_ethnicity
  - ethnicity.outliers
  - ethnicity.population_file_outliers
  - All the figures (they might be mirrored).
- 2_sex_check
  - sexcheck.list_problem_sex
  - sexcheck.list_problem_sex_ids

CHAPTER

THREE

INPUT FILES

To use *pyGenClean*, two sets of files are required: a set of genotype files and a configuration file.

3.1 Genotype Files

The input files of the main program (`run_pyGenClean`) is one of the following:

- PLINK's pedfile format (use *pyGenClean*'s `--file` option) consist of two files with the following extensions: `PED` and `MAP`.
- PLINK's transposed pedfile format (use *pyGenClean*'s `--tfile` option) consist of two files with the following extensions: `TPED` and `TFAM`.
- PLINK's binary pedfile format (use *pyGenClean*'s `--bfile` option) consist of three files with the following extensions: `BED`, `BIM` and `FAM`.

For more information about these file formats, have a look at PLINK's website, in the *Basic usage/data formats* section (http://pngu.mgh.harvard.edu/~purcell/plink/data.shtml).

Warning: If the format used is the *transposed* one, the columns **must** be separated using **tabulations**, but alleles of each markers need to be separated by a single space.
To create this exact transposed pedfile format, you need to use the following PLINK's options:

- `--recode` to recode the file.
- `--transposed` to create an output file in the transposed pedfile format.
- `--tab` to use tabulations.

3.2 Configuration File

To customized *pyGenClean*, a basic configuration file is required. It tells which script to use in a specific order. It also sets the different options and input files, so that the analysis is easy to replicate or modify.

The configuration file consists of sections, led by a `[section]` header (contiguous integers which gives the order of the pipeline) and followed by customization of this particular part of the pipeline. Lines preceded by a # are comments and are not read by *pyGenClean*.

The following example first removes samples with a missing rate of 10% and more, then removes markers with a missing rate of 2% and more. Finally, it removes the samples with a missing rate of 2% and more.

```
[1]
# Removes sample with a missing rate higher than 10%.
script = sample_missingness
mind = 0.1

[2]
# Removes markers with a missing rate higher than 2%.
script = snp_missingness
geno = 0.02

[3]
# Removes sample with a missing rate higher than 2%.
script = sample_missingness
mind = 0.02
```

For a more thorough example, complete configuration files are available for download at http://www.statgen.org and are explained in the *Configuration Files* section. For a list of available modules and standalone script, refer to the *List of Modules and their Options*.

3.2.1 List of Modules and their Options

The following sections show a list the available scripts that can be used in the configuration file, along with their options for customization.

Duplicated Samples

The name to use in the configuration file is `duplicated_samples` and the *List of options for the duplicated_samples script.* table shows its configuration.

Table 3.1: List of options for the duplicated_samples script.

Option	Type	Description
`--sample-completion-threshold`	`FLOAT`	The completion threshold to consider a replicate when choosing the best replicates and for creating the composite samples. [default: 0.9]
`--sample-concordance-threshold`	`FLOAT`	The concordance threshold to consider a replicate when choosing the best replicates and for creating the composite samples. [default: 0.97]

The name of the standalone script is `pyGenClean_duplicated_samples`.

Duplicated Markers

The name to use in the configuration file is `duplicated_snps` and the *List of options for the duplicated_snps script.* table shows its configuration.

Table 3.2: List of options for the duplicated_snps script.

Option	Type	Description
`--snp-completion-threshold`	`FLOAT`	The completion threshold to consider a replicate when choosing the best replicates and for composite creation. [default: 0.9]
`--snp-concordance-threshold`	`FLOAT`	The concordance threshold to consider a replicate when choosing the best replicates and for composite creation. [default: 0.98]
`--frequency_difference`	`FLOAT`	The maximum difference in frequency between duplicated markers [default: 0.05]

The name of the standalone script is `pyGenClean_duplicated_snps`.

Clean No Call and Only Heterozygous Markers

The name to use in the configuration file is `noCall_hetero_snps` and there are no customization possible.

The name of the standalone script is `pyGenClean_clean_noCall_hetero_snps`.

Sample Missingness

The name to use in the configuration file is `sample_missingness` and the *List of options for the sample_missingness script.* table shows its configuration.

Table 3.3: List of options for the sample_missingness script.

Option	Type	Description
`--mind`	`FLOAT`	The missingness threshold (remove samples with more than x percent missing genotypes). [Default: 0.100]

The name of the standalone script is `pyGenClean_sample_missingness`.

Marker Missingness

The name to use in the configuration file is `snp_missingness` and the *List of options for the snp_missingness script.* table shows its configuration.

Table 3.4: List of options for the snp_missingness script.

Option	Type	Description
`--geno`	`FLOAT`	The missingness threshold (remove SNPs with more than x percent missing genotypes). [Default: 0.020]

The name of the standalone script is `pyGenClean_snp_missingness`.

Sex Check

The name to use in the configuration file is `sex_check` and the *List of options for the sex_check script.* table shows its configuration.

Table 3.5: List of options for the sex_check script.

Option	Type	Description
`--femaleF`	`FLOAT`	The female F threshold. [default: < 0.300000]
`--maleF`	`FLOAT`	The male F threshold. [default: > 0.700000]
`--nbChr23`	`INT`	The minimum number of markers on chromosome 23 before computing Plink's sex check [default: 50]
`--gender-plot`		Create the gender plot (summarized chr Y intensities in function of summarized chr X intensities) for problematic samples. Not used by default.
`--sex-chr-intensities`	`FILE`	A file containing alleles intensities for each of the markers located on the X and Y chromosome for the gender plot.
`--gender-plot-format`	`STRING`	The output file format for the gender plot (png, ps, or pdf formats are available). [default: png]
`--lrr-baf`		Create the LRR and BAF plot for problematic samples. Not used by default.
`--lrr-baf-raw-dir`	`DIR`	Directory or list of directories containing information about every samples (BAF and LRR).
`--lrr-baf-format`	`STRING`	The output file format for the LRR and BAF plot (png, ps or pdf formats are available). [default: png]

The name of the standalone script is `pyGenClean_sex_check`. If you want to redo the BAF and LRR plot or the gender plot, you can use the `pyGenClean_baf_lrr_plot` and `pyGenClean_gender_plot` scripts, respectively.

Plate Bias

The name to use in the configuration file is `plate_bias` and the *List of options for the plate_bias script.* table shows its configuration.

Table 3.6: List of options for the plate_bias script.

Option	Type	Description
`--loop-assoc`	`FILE`	The file containing the plate organization of each samples. Must contains three column (with no header): famID, indID and plateName.
`--pfilter`	`FLOAT`	The significance threshold used for the plate effect. [default: 1.0e-07]

The name of the standalone script is `pyGenClean_plate_bias`.

Heterozygous Haploid

The name to use in the configuration file is `remove_heterozygous_haploid` and there are no customization possible.

The name of the standalone script is `pyGenClean_remove_heterozygous_haploid`.

Related Samples

The name to use in the configuration file is `find_related_samples` and the *List of options for the find_related_samples script.* table shows its configuration.

Table 3.7: List of options for the find_related_samples script.

Option	Type	Description
`--genome-only`		Only create the genome file. Not selected by default.
`--min-nb-snp`	`INT`	The minimum number of markers needed to compute IBS values. [Default: 10000]
`--indep-pairwise`	`INT INT FLOAT`	Three numbers: window size, window shift and the r2 threshold. [default: ['50', '5', '0.1']]
`--maf`	`FLOAT`	Restrict to SNPs with MAF >= threshold. [default: 0.05]
`--ibs2-ratio`	`FLOAT`	The initial IBS2* ratio (the minimum value to show in the plot. [default: 0.8]
`--sge`		Use SGE for parallelization.
`--line-per-file-for-sge`	`INT`	The number of line per file for SGE task array. [default: 100]

The name of the standalone script is `pyGenClean_find_related_samples`. Even though randomly choosing a subset of related samples is done automatically, you can use the `pyGenClean_merge_related_samples` to perform it again.

Ethnicity

The name to use in the configuration file is `check_ethnicity` and the *List of options for the check_ethnicity script.* table shows its configuration.

Table 3.8: List of options for the check_ethnicity script.

Option	Type	Description
--ceu-bfile	FILE	The input file prefix (will find the plink binary files by appending the prefix to the .bim, .bed and .fam files, respectively.) for the CEU population.
--yri-bfile	FILE	The input file prefix (will find the plink binary files by appending the prefix to the .bim, .bed and .fam files, respectively.) for the CEU population.
--jpt-chb-bfile	FILE	The input file prefix (will find the plink binary files by appending the prefix to the .bim, .bed and .fam files, respectively.) for the JPT-CHB population.
--min-nb-snp	FILE	The minimum number of markers needed to compute IBS values. [Default: 8000]
--indep-pairwise	INT INT FLOAT	Three numbers: window size, window shift and the r2 threshold. [default: ['50', '5', '0.1']]
--maf	INT	Restrict to SNPs with MAF >= threshold. [default: 0.05]
--sge		Use SGE for parallelization.
--line-per-file-for-sge	INT	The number of line per file for SGE task array. [default: 100]
--nb-components	INT	The number of component to compute. [default: 10]
--outliers-of	STRING	Finds the outliers of this population. [default: CEU]
--multiplier	FLOAT	To find the outliers, we look for more than x times the cluster standard deviation. [default: 1.9]
--xaxis	STRING	The component to use for the X axis. [default: C1]
--yaxis	STRING	The component to use for the Y axis. [default: C2]
--format	STRING	The output file format (png, ps, pdf, or X11 formats are available). [default: png]
--title	STRING	The title of the MDS plot. [default: C2 in function of C1 - MDS]
--xlabel	STRING	The label of the X axis. [default: C1]
--ylabel	STRING	The label of the Y axis. [default: C2]

The name of the standalone script is `pyGenClean_check_ethnicity`. If you want to redo the outlier detection using a different multiplier, have a look at the `pyGenClean_find_outliers` script. If you want to redo any MDS plot, have a look at the `pyGenClean_plot_MDS` script.

Minor Allele Frequency of Zero

The name to use in the configuration file is `flag_maf_zero` and there are no customization possible.

The name of the standalone script is `pyGenClean_flag_maf_zero`.

Hardy Weinberg Equilibrium

The name to use in the configuration file is `flag_hw` and the *List of options for the flag_hw script.* table shows its configuration.

Table 3.9: List of options for the flag_hw script.

Option	Type	Description
`--hwe`	`FLOAT`	The Hardy-Weinberg equilibrium threshold. [default: 1e-4]

The name of the standalone script is `pyGenClean_flag_hw`.

Subsetting the Data

The name to use in the configuration file is `subset` and the *List of options for the subset script.* table shows its configuration.

Table 3.10: List of options for the subset script.

Option	Type	Description
`--exclude`	`FILE`	A file containing SNPs to exclude from the data set.
`--extract`	`FILE`	A file containing SNPs to extract from the data set.
`--remove`	`FILE`	A file containing samples (FID and IID) to remove from the data set.
`--keep`	`FILE`	A file containing samples (FID and IID) to keep from the data set.

The name of the standalone script is `pyGenClean_subset_data`.

Comparison with a Gold Standard

The name to use in the configuration file is `compare_gold_standard` and the *List of options for the compare_gold_standard script.* table shows its configuration.

Table 3.11: List of options for the compare_gold_standard script.

Option	Type	Description
`--gold-bfile`	`FILE`	The input file prefix (will find the plink binary files by appending the prefix to the .bim, .bed and .fam files, respectively.) for the Gold Standard .
`--same-samples`	`FILE`	A file containing samples which are present in both the gold standard and the source panel. One line by identity and tab separated. For each row, first sample is Gold Standard, second is source panel.
`--source-manifest`	`FILE`	The illumina marker manifest.
`--source-alleles`	`FILE`	A file containing the source alleles (TOP). Two columns (separated by tabulation, one with the marker name, the other with the alleles (separated by space). No header.
`--sge`		Use SGE for parallelization.
`--do-not-flip`		Do not flip SNPs. WARNING: only use this option only if the Gold Standard was generated using the same chip (hence, flipping is unnecessary).
`--use-marker-names`		Use marker names instead of (chr, position). WARNING: only use this options only if the Gold Standard was generated using the same chip (hence, they have the same marker names).

The name of the standalone script is `pyGenClean_compare_gold_standard`.

CHAPTER

FOUR

INFORMATION ABOUT THE PROTOCOL

The following sections describe the proposed protocol and provides information about which file should be looked for quality control. Finally, configuration files (with all available parameters) are given.

4.1 Proposed Protocol

4.1.1 Preprocessing Steps

- Remove SNPs without chromosomal and physical position (chromosome and position of 0).
- Remove INDELs (markers with alleles `I` or `D`).
- Determine if there are duplicated samples. These samples must have exactly the same family (`FID`) and individual (`IID`) identification to be treated as duplicated samples by the *Duplicated Samples Module*. PLINK's option `--update-ids` could be used.
- If input is a transposed pedfile, be sure to use PLINK's option `--tab` to produce the appropriate file format.
- For the *Plate Bias Module*, a text file explaining the plate distribution of each sample must be provided using the option `--loop-assoc` in the configuration file. The following columns are required (in order, without a header):
  - the family identification;
  - the individual identification;
  - and the plate identification.
- Produce parameter files (see the *Configuration Files* for details about parameter file).
- To launch the analysis consult the section *How to Run the Pipeline*.

4.1.2 Duplicated Samples Module

Note: Input files:

- PLINK transposed pedfiles from the *Preprocessing Steps*.

---

1. Examine the log to confirm options used and to detect any problems occurring while running the script.
2. Examine `dup_samples.diff` to evaluate if some samples have many discordant genotypes (this could indicate a possible samples mix up). To identify discordant samples, use the following command line:

```
$ cut -f4 dup_samples.diff | sort -k1,1 | uniq -c
$ cut -f5 dup_samples.diff | sort -k1,1 | uniq -c
```

If samples are present more than 10,000 times (for 2.5E-6 SNPs) this could indicate a sample mix up.

3. Examine `dup_samples.not_good_enough` to determine if samples have a concordance rate below the threshold set by the user. These samples **are present** in the `dup_samples.final.tfam` if they are the chosen ones.
4. Examine `dup_samples.summary` to evaluate completion rate and concordance between the replicates of potentially problematic samples.
5. Examine `dup_samples.concordance` file for the problematic samples; this could help to determine which sample is the discordant replicate.
6. If a sample appears problematic rename it and keep it in the analysis to determine if it is a duplicate of another sample (mix up) with the related sample module.

If necessary, samples present in the `dup_samples.not_good_enough` file can be removed from the data set with the subset module (see the *First Subset Module (optional)*). If not, proceed to the *Duplicated Markers Module*).

4.1.3 First Subset Module (optional)

Note: Input files:

From the *Duplicated Samples Module*:

- `dup_samples.final.tfam`
- `dup_samples.final.tped`

---

1. Extract the family (`FID`) and individual (`IID`) identification from `dup_samples.not_good_enough` with the following command line:

```
$ cut -f3,4 dup_samples.not_good_enough | sed "1d" | sort -k1,1 \
>     | uniq > samples_to_remove
```

4.1.4 Duplicated Markers Module

Note: Input files:

From the *Duplicated Samples Module*:

- `dup_samples.final.tfam`
- `dup_samples.final.tped`

or from the *First Subset Module (optional)*:

- `subset.bed`
- `subset.bim`
- `subset.fam`

---

1. Examine the log to confirm options used and to detect any problems occurring while running the script

2. Examine `dup_snps.duplicated_marker_names` to detect SNPs with exactly the same name but mapping to different chromosomal location. (This file is not produce if no duplicated marker names are identified).
3. Determine the number of duplicated SNPs merged (same allele, same frequency, etc). SNPs merged were removed and are listed in the file `dup_snps.removed_duplicates`. Number of lines in this file corresponds to number of SNPs merged. SNPs not merged and reasons why (*e.g.* `homo_hetero`, `diff_frequency`, `homo_flip`, etc.) are present in file `dup_snps.problems`.
4. SNPs with concordance rate below the threshold are present in `dup_snps.not_good_enough`. To have the list of those SNPs:

```
$ grep -w concordance dup_snps.not_good_enough | cut -f1 \
>     > SNP_with_low_concordance_rate
```

If necessary, use the subset option in the configuration file to remove the low concordance rate SNPs (see the *Second Subset Module (optional)*).

4.1.5 Second Subset Module (optional)

Note: Input files:

From the *Duplicated Markers Module*:

- `dup_snps.final.tfam`
- `dup_snps.final.tped`

- Extract SNPs with concordance rate below the threshold set by the user with the command line

```
$ grep -w concordance dup_snps.not_good_enough | cut -f1 \
>     > SNP_with_low_concordance_rate
```

4.1.6 Clean No Call and Only Heterozygous Markers Module

Note: Input files:

From the *Duplicated Markers Module*:

- `dup_snps.final.tfam`
- `dup_snps.final.tped`

or from the *Second Subset Module (optional)*:

- `subset.bed`
- `subset.bim`
- `subset.fam`

---

1. Examine the log to confirm options used and to detect any problems occurring while running the script.
2. SNPs removed because they are failed are listed in `clean_noCall_hetero.allFailed`.
3. SNPs removed because they are all heterozygous are listed in `clean_noCall_hetero.allHetero`.

4.1.7 Sample Missingness Module (mind 0.1)

Note: Input files:

From the *Clean No Call and Only Heterozygous Markers Module*:

- `clean_noCall_hetero.tfam`
- `clean_noCall_hetero.tped`

1. Examine the log to confirm options used and to detect any problems occurring while running the script.
2. Examine PLINK's log file to detect any problem at this step.
3. Individuals removed because they did not pass the completion rate threshold are listed in `clean_mind.irem`.

4.1.8 Marker Missingness Module

Note: Input files:

From the *Sample Missingness Module (mind 0.1)*:

- `clean_mind.bed`
- `clean_mind.bim`
- `clean_mind.fam`

1. Examine the log to confirm options used and to detect any problems occurring while running the script.
2. Examine PLINK's log file to detect any problem at this step.
3. SNPs removed because they did not pass the completion rate threshold are listed in `clean_geno.removed_snps`.

4.1.9 Sample Missingness Module (mind 0.02)

Note: Input files:

From the *Marker Missingness Module*:

- `clean_geno.bed`
- `clean_geno.bim`
- `clean_geno.fam`

1. Examine the log to confirm options used and to detect any problems occurring while running the script.
2. Examine PLINK's log file to detect any problem at this step.
3. Individuals removed because they did not pass the completion rate threshold are listed in `clean_mind.irem`.

4.1.10 Sex Check Module

Note: Input files:

From *Sample Missingness Module (mind 0.02)*:

- `clean_mind.bed`
- `clean_mind.bim`
- `clean_mind.fam`

1. Examine the log to confirm options used and to detect any problems occurring while running the script.
2. Examine PLINK's log file to detect any problem at this step.
3. Examine `sexcheck.list_problem_sex`, it contains all individuals identified by PLINK as having gender problem.
4. Examine `sexcheck.chr23_recodeA.raw.hetero` to determine heterozygosity on the X chromosome of problematic samples. Consanguineous females may have low heterozygosity on the X chromosome. If many genotyped SNPs are rare, heterozygosity may also be low.
5. Examine `sexcheck.chr24_recodeA.raw.noCall` to determine the number of Y markers with missing calls. Females have low number of genotypes for Y chromosome markers (high values of missing calls), but is often not equal to 0 probably because some Y markers come from pseudo autosomal regions. Column `nbGeno` is the total number of genotypes check and `nbNoCall` is the number of genotypes with missing calls on chromosome Y. Males should have low values in this column while females have higher number of missing calls but not equal to the total number of genotypes tested.

If probe intensities from X and Y chromosomes are available and the gender plot has been created:

1. Examine the log to confirm options used and to detect any problems occurring while running the script.
2. Examine `sexcheck.png` to detect any individuals in the XXY or X0 regions, females in the male cluster and males in the female cluster (see the *Gender plot* figure). Confirm if possible the gender problems identified with the previous sex check problem step.

If intensities file for each sample are available and the BAF and LRR plot has been created:

1. Examine the log to confirm options used and to detect any problems occurring while running the script.
2. Examine `sexcheck_sample-id_lrr_baf.png` for each sample. Usually, females have LRR values around 0 (between -0.5 and 0.5) while males have LRR values between -0.5 and -1. Females have three lines on BAF graphics: one at 1 (homozygous for the B allele), one at 0.5 (heterozygous AB) and one at 0 (homozygous for the A allele). Males have two lines: one at 1 (homozygous for the B allele) and one a at 0 (homozygous for the A allele). For more details, see the *The Plots* section of the *Sex Check Module*.

Keep individuals identified with gender problem until the *Related Samples Module* (mix up of samples could be resolved at this step).

4.1.11 Plate Bias Module

Note: Input files:

From the *Sample Missingness Module (mind 0.02)*:

- `clean_mind.bed`
- `clean_mind.bim`

- clean_mind.fam

or if subset option is used to remove SNPs from nof file (see below):

- subset.bed
- subset.bim
- subset.fam

---

1. Verify if there is a nof file produce by PLINK when the input files for this step were produced (from the *Sample Missingness Module (mind 0.02)*). The nof contains SNPs with no founder genotypes observed. If so, remove the SNPs present in the nof file using the subset tool before launching the plate bias analysis. Those SNPs, if they are not removed will produced an error message when PLINK performs the loop-assoc analysis and the following message will be present in PLINK's log file plate_bias.log: "ERROR: FEXACT error 3". SNPs on chromosome 24 could also produce this error.
2. Examine the log to confirm options used and to detect any problems occurring while running the script.
3. Examine plate_bias.log to detect any problem at this step.
4. The plate_bias.significant_SNPs.txt file contains a list of SNPs with P value below the threshold. Care should be taken with those SNPs if significant results are obtained in association tests. These SNPs are NOT removed from the data set, only flagged.
5. Low MAF can explain part of plate bias. Examine the output file plate_bias.significant_SNPs.frq to determine if SNPs have low MAF. Other reasons explaining plate bias are relatedness or ethnicity of individuals assign to the same plates and none of them on other plates.

4.1.12 Related Samples Module

Note: Input files:

From the *Sample Missingness Module (mind 0.02)*:

- clean_mind.bed
- clean_mind.bim
- clean_mind.fam

---

1. Examine the log to confirm options used and to detect any problems occurring while running the script.
2. File ibs.pruning_0.1.prune.in contains the list of uncorrelated SNPs used for the IBS analysis
3. Examine ibs.related_individuals_z1.png and ibs.related_individuals_z2.png to detect if there are samples in the parent-child, duplicated samples, first degree relative and second degree relative areas. (see *Z1 in function of IBS2 ratio* and *Z2 in function of IBS2 ratio* plots).
4. File ibs.related_individuals lists pairs of related individuals. Index column indicates group of related samples. Status column indicated the probable link between pair of individuals based on Z_0, Z_1 and Z_2 values (see the *IBD allele sharing values* table [for which Z values are approximation] or Step9.find_related_samples.extractRelatedIndividuals() function for thresholds).
5. If there are known duplicated samples, examine ibs.related_individuals to determine if they were identified correctly, if not this could indicate a possible samples mix up.
6. File ibs.choosen_related_individuals contains a list of related samples to keep. One related sample from the pair is randomly selected. If there are a group of related individuals, one sample in randomly selected

from the group. All non selected samples are listed in `ibs.discarded_related_individuals` and should be removed from the analysis at a later stage.

Table 4.1: IBD allele sharing values

Relationship	k_0	k_1	k_2	**Coancestry** $\theta = 1/2k_2 + 1/4k_1$
Unrelated	1	0	0	0
Identical twins	0	0	1	$1/2$
Parent-child	0	1	0	$1/4$
Full siblings	$1/4$	$1/2$	$1/4$	$1/4$
Half siblings	$1/2$	$1/2$	0	$1/8$
Uncle nephew	$1/2$	$1/2$	0	$1/8$
Grandparent grandchild	$1/2$	$1/2$	0	$1/8$
Double first cousins	$9/16$	$3/8$	$1/16$	$1/8$
First cousins	$3/4$	$1/4$	0	$1/16$

4.1.13 Ethnicity Module

Note: Input files:

From the *Sample Missingness Module (mind 0.02)*:

- `clean_mind.bed`
- `clean_mind.bim`
- `clean_mind.fam`

1. Examine the log to confirm options used and to detect any problems occurring while running the script.
2. File `ethnic.ibs.pruning_0.1.prune.in` contains the list of uncorrelated SNPs used for the MDS analysis.
3. File `ethnic.mds.mds` contains the list of principale components as calculated by PLINK.
4. Examine `ethnicity.mds.png`, `ethnicity.before.png`, `ethnicity.after.png` and `ethnicity.outliers.png` to detect samples outside the selected cluster (see the generated *The Plots* from the *Ethnicity Module* for more information).

   If there are too many outliers still present in the data set (*i.e.* radius is too large), analysis can be redone using the `pyGenClean_find_outliers` standalone script, using a different value for `--multiplier`. For more information, refer to the *Finding Outliers* section of the *Ethnicity Module*.
5. Samples outside the selected cluster are listed in `ethnicity.outliers`. If necessary those samples could be removed at a later stage with the subset option.

4.1.14 Third Subset Module

Note: Input files:

From the *Sample Missingness Module (mind 0.02)*:

- `clean_mind.bed`
- `clean_mind.bim`

- `clean_mind.fam`

Use the subset module to remove samples with gender problems (the *Sex Check Module*), outliers from the ethnicity cluster (the *Ethnicity Module*), related samples (the *Related Samples Module*) and any other samples that need to be removed from the data set.

- To produces a file containing all the samples to remove from the dataset:

```
$ cat sexcheck.list_problem_sex_ids ibs.discarded_related_individuals \
>     ethnicity.outliers > samples_to_remove.txt
```

One sample may be removed for more than one reason, hence be present more than one time in the final `samples_to_remove.txt` file. This is not an issue for this step.

4.1.15 Heterozygote Haploid Module

Note: Input files:

From the *Third Subset Module*:

- `subset.bed`
- `subset.bim`
- `subset.fam`

Samples with gender problems **must have been removed before** performing this module.

1. Examine the log to confirm options used and to detect any problems occurring while running the script.
2. Examine `without_hh_genotypes.log` to detect any problem at this step.

Number of heterozygous haploid genotypes set to missing are indicated in `without_hh_genotypes.log` file.

4.1.16 Minor Allele Frequency of Zero Module

Note: Input files:

From the *Heterozygote Haploid Module*:

- `without_hh_genotypes.bed`
- `without_hh_genotypes.bim`
- `without_hh_genotypes.fam`

---

1. Examine the log to confirm options used and to detect any problems occurring while running the script.
2. Examine `flag_maf_0.log` to detect any problem at this step.
3. The file `flag_maf_0.na_list` contains a list of SNPs with minor allele frequency of 0.

If necessary, use subset module to remove SNPs with minor allele frequency of 0, since they were only flagged using the *Fourth Subset Module (optional)*.

4.1.17 Fourth Subset Module (optional)

Note: Input files:

From the *Heterozygote Haploid Module*:

- `without_hh_genotypes.bed`
- `without_hh_genotypes.bim`
- `without_hh_genotypes.fam`

1. Examine the log to confirm options used and to detect any problems occurring while running the script.
2. Examine `subset.log` to detect any problem at this step.

4.1.18 Hardy Weinberg Equilibrium Module

Note: Input files:

From the *Heterozygote Haploid Module*:

- `without_hh_genotypes.bed`
- `without_hh_genotypes.bim`
- `without_hh_genotypes.fam`

or from the *Fourth Subset Module (optional)*:

- `subset.bed`
- `subset.bim`
- `subset.fam`

1. Examine the log to confirm options used and to detect any problems occurring while running the script.
2. Examine `flag_hw.threshold_1e-4.log` and `flag_hw.threshold_Bonferroni.log` to detect any problem at this step.
3. The files `flag_hw.snp_flag_threshold_Bonferroni` and `flag_hw.snp_flag_threshold_1e-4` contain lists of SNPs with P value below Bonferroni and below 1×10^{-4} threshold, respectively.

The markers are only flagged using this module. If you want to remove those markers, have a look at the *Fifth Subset Module (optional)*.

4.1.19 Fifth Subset Module (optional)

Note: Input files:

From the *Heterozygote Haploid Module*:

- `without_hh_genotypes.bed`
- `without_hh_genotypes.bim`
- `without_hh_genotypes.fam`

or from the *Fourth Subset Module (optional)*:

- `subset.bed`
- `subset.bim`
- `subsert.fam`

1. Examine the log to confirm options used and to detect any problems occurring while running the script.
2. Examine `subset.log` to detect any problem at this step.

4.2 Configuration Files

Two default configuration files are available to run the proposed protocol. Before using them, be sure to follow the *Preprocessing Steps* described in the *Proposed Protocol* section.

Note that lines starting with a `#` are comments, and are not used by the pipeline. The default parameters were commented out, and could be uncommented to change their values.

4.2.1 First Configuration File

This file should be use with the original dataset as input. Only change the `loop-assoc` file name in the `plate_bias` section (`[8]`) and the reference population files (`ceu-bfile`, `yri-bfile` and `jpt-chb-bfile` in the `check_ethnicity` section (`[10]`). Those last three datasets are provided and can be downloaded at http://www.statgen.org.

If you want to generate the gender and BAF and LRR plots, you will require to provide the intensities (`sex-chr-intensities` and `lrr-baf-raw-dir` in the `sex_check` section (`[7]`) after uncommenting the required options).

```
# This is the first part of example configuration files for performing efficient
# data clean up. All commented out parameters are those that are used by
# default.


[1]
# ##############################################################################
# Checks missing rate and pairwise concordance of duplicated samples. Duplicated
# samples should have same family and individual identification numbers. The
# names can be modified directly in the transposed pedfile.
# ##############################################################################

script = duplicated_samples
# sample-completion-threshold = 0.9
# sample-concordance-threshold = 0.97



[2]
# ##############################################################################
# Checks missing rate and pairwise concordance of duplicated markers. Duplicated
# markers are found by looking at their chromosomal position. No modification of
# the transposed bedfile is required.
# ##############################################################################

script = duplicated_snps
# snp-completion-threshold = 0.9
```

```
# snp-concordance-threshold = 0.98
# frequency_difference = 0.05



[3]
# ##############################################################################
# Finds and removes markers which have a missing rate of 100% or markers (not
# located on mitochondrial chromosome) that have a heterozygosity rate of 0%.
# ##############################################################################

script = noCall_hetero_snps



[4]
# ##############################################################################
# Removes sample with a missing rate higher than a user defined threshold. For
# this step, we recommend using a threshold of 10% missing rate as samples with
# a missing rate of 2% will be later removed.
# ##############################################################################

script = sample_missingness
# mind = 0.1



[5]
# ##############################################################################
# Removes markers with a missing rate higher than a user defined threshold. For
# this step, we recommend using a threshold of 2% missing rate.
# ##############################################################################

script = snp_missingness
# geno = 0.02



[6]
# ##############################################################################
# Removes sample with a missing rate higher than a user defined threshold. For
# this step, we recommend using a threshold of 2% missing rate.
# ##############################################################################

script = sample_missingness
mind = 0.02



[7]
# ##############################################################################
# Using PLINK, finds samples with gender issues, according to heterozygosity
# rate on the X chromosome. If you want to produce a gender plot, you need to
# uncomment the "gender-plot" option and provide a file containing marker
# intensities on the X and Y chromosomes. If you want to produce a BAF and LRR
# plot, you need to uncomment the "lrr-baf" option and provide a directory
# containing the BAF and LRR values of each marker on the X and Y chromosomes
# (one file per sample).
```

```
# ##############################################################################

script = sex_check
# femaleF = 0.3
# maleF = 0.7
# nbChr23 = 50
# gender-plot
# sex-chr-intensities = /PATH/TO/FILE/CONTAINING/INTENSITIES_FILE.txt
# gender-plot-format = png
# lrr-baf
# lrr-baf-raw-dir = /PATH/TO/DIRECTORY/CONTAINING/BAF_LRR_FILES.txt
# lrr-baf-format = png



[8]
# ##############################################################################
# Using PLINK, performs a plate bias analysis, using a p value threshold of
# 1.0e-7.
# ##############################################################################

script = plate_bias
loop-assoc = /PATH/TO/FILE/CONTAINING/PLATE_INFORMATION.txt
# pfilter = 1.0e-07



[9]
# ##############################################################################
# Checks for related individual and randomly keeps one of each related group. If
# you have a server with a DRMAA-compliant distributed resource management
# system, you can uncomment the "sge" and the "line-per-file-for-sge" options,
# to run this step in parallel.
# ##############################################################################

script = find_related_samples
# min-nb-snp = 10000
# indep-pairwise = 50 5 0.1
# maf = 0.05
# ibs2-ratio = 0.8
# sge
# line-per-file-for-sge = 100



[10]
# ##############################################################################
# Using PLINK, computes the MDS value of each sample, and using three reference
# populations (CEU, YRI and JPT-CHB), finds outliers of one of those three
# reference population. You might need to change the "multiplier" option to be
# more or less stringent, according to you dataset. If you have a server with a
# DRMAA-compliant distributed resource management system, you can uncomment the
# "sge" and the "line-per-file-for-sge" options, to run this step in parallel.
# ##############################################################################

script = check_ethnicity
ceu-bfile = /PATH/TO/PLINK/BINARY/FILE/FOR/CEU_population
yri-bfile = /PATH/TO/PLINK/BINARY/FILE/FOR/YRI_population
```

```
jpt-chb-bfile = /PATH/TO/PLINK/BINARY/FILE/FOR/JPT-CHB_population
# min-nb-snp = 8000
# indep-pairwise = 50 5 0.1
# maf = 0.05
# sge
# line-per-file-for-sge = 100
# nb-components = 10
# outliers-of = CEU
# multiplier = 1.9
# xaxis = C1
# yaxis = C2
# format = png
# title = "C2 in function of C1 - MDS"
# xlabel = C1
# ylabel = C2
```

4.2.2 Second Configuration File

This configuration file should be run after the *First Configuration File* and with the output of the second sample missingness section (`[6]` in the *First Configuration File*).

A file containing the samples and markers to be removed should be created using the output of the `sex_check`, `find_related_samples`, `check_ethnicity` and `plate_bias` sections of the *First Configuration File*.

```
# This is the second part of example configuration files for performing
# efficient data clean up. All commented out parameters are those that are used
# by default.

# The input file should be the output file of the second sample missigness step
# (which is the one that has been used by any of these scripts):
#     - sex_check
#     - find_related_samples
#     - check_ethnicity
#     - plate_bias

# Note that the final usable dataset is the one located in the directory where
# "remove_heterozygous_haploid" was run (which is the one that has been used by
# any of these scripts):
#     - flag_maf_zero
#     - flag_hw
# Hence, if you want to remove the flagged markers, you should use
# pyGenClean_subset_data on markers in the "flag_maf_zero" and "flag_hw"
# directories using the PLINK's binary file located in
# "remove_heterozygous_haploid".

[11]
# ##############################################################################
# After manually checking that everything went fine in the previous steps, you
# need to create a list of samples to remove from steps [7] to [10] and a list
# of markers to exclude from steps [6]. Just create a file containing family and
# individual identification numbers for all those samples to remove.
# ##############################################################################

script = subset
remove = /PATH/TO/FILE/CONTAINING/ALL_SAMPLES_FROM_PREVIOUS_STEPS_TO_REMOVE.txt
exclude = /PATH/TO/FILE/CONTAINING/ALL_MARKERS_FROM_PREVIOUS_STEPS_TO_EXCLUDE.txt

```

```
34
35
36  [12]
37  # ##############################################################################
38  # Removes heterozygous haploid genotypes from the dataset.
39  # ##############################################################################
40
41  script = remove_heterozygous_haploid
42
43
44
45  [13]
46  # ##############################################################################
47  # Flags uninformative markers (with a MAF of 0). This step only flag markers.
48  # You might want to exclude them later on.
49  # ##############################################################################
50
51  script = flag_maf_zero
52
53
54
55  [14]
56  # ##############################################################################
57  # Flags markers that fail HWE test for a p value of 1e-4 and after Bonferroni
58  # correction. This step only flag markers. You might want to exclude them later
59  # on.
60  # ##############################################################################
61
62  script = flag_hw
63  # hwe = 1e-4
```

4.3 How to Run the Pipeline

Warning: If you are working in a Linux environment, before doing anything, be sure to activate the Python virtual environment (refer to the *Python Virtual Environment* installation section for more information).
If you are working in a Windows environment, you will need to modify the command so that it loads correctly. For example, the following command

```
$ run_pyGenClean \
>     --conf conf_1.txt \
>     --tfile /PATH/TO/ORIGINAL/DATASET_PREFIX
```

will become

```
> python C:\Python27\Scripts\run_pyGenClean ^
    --conf conf_1.txt ^
    --bfile \PATH\TO\ORIGINAL\DATASET_PREFIX
```

Modify the *First Configuration File* so that it suits your needs. After following the *Preprocessing Steps* described in the *Proposed Protocol* section, run the following command:

```
$ run_pyGenClean \
>     --conf conf_1.txt \
>     --tfile /PATH/TO/ORIGINAL/DATASET_PREFIX
```

While the protocol is running, check the outputs according to the *Proposed Protocol*. If there are any problems, interrupt the analysis and make the required modifications. The completed steps can be skipped by commenting them out, while using the last output dataset as the input one for the steps that need to be done again.

Once everything was checked, locate the samples and the markers that need to be removed. For example, if the output directory from the first dataset is `data_clean_up.YYYY-MM-DD_HH.MM.SS`, the following command will help you:

```
$ output_dir=data_clean_up.YYYY-MM-DD_HH.MM.SS
$ cat $output_dir/7_sex_check/sexcheck.list_problem_sex_ids \
>     $output_dir/9_find_related_samples/ibs.discarded_related_individuals \
>     $output_dir/10_check_ethnicity/ethnicity.outliers \
>     > samples_to_remove.txt
```

Then, modify the first `subset` section in the *Second Configuration File* so that it reads:

```
[11]
script = subset
remove = samples_to_remove.txt
exclude = data_clean_up.YYYY-MM-DD_HH.MM.SS/8_plate_bias/plate_bias.significant_SNPs.txt
```

Once everything was checked, run the following command to finish the data clean up pipeline:

```
$ output_dir=data_clean_up.YYYY-MM-DD_HH.MM.SS
$ run_pyGenClean \
>     --conf conf_2.txt \
>     --bfile $output_dir/6_sample_missingness/clean_mind
```

If you want to removed the markers that were flagged in the `flag_maf_zero` and `flag_hw` section, performed the following commands (using the newly created output directory `data_clean_up.YYYY-MM-DD_HH.MM.SS`):

```
$ output_dir=data_clean_up.YYYY-MM-DD_HH.MM.SS
$ cat $output_dir/13_flag_maf_zero/flag_maf_0.list \
>     $output_dir/14_flag_hw/flag_hw.snp_flag_threshold_1e-4 \
>     > markers_to_exclude.txt
$ pyGenClean_subset_data \
>     --ifile $output_dir/14_remove_heterozygous_haploid/without_hh_genotypes \
>     --is-bfile \
>     --exclude markers_to_exclude.txt \
>     --out final_dataset
```

CHAPTER

FIVE

THE ALGORITHM

All the functions used for this project are shown and explained in the following section:

5.1 The Data Clean Up Module

exception run_data_clean_up.**ProgramError**(*msg*)
An Exception raised in case of a problem.

Parameters **msg** (*string*) – the message to print to the user before exiting.

run_data_clean_up.**all_files_exist**(*file_list*)
Check if all files exist.

Parameters **file_list** (*list of strings*) – the names of files to check.

Returns True if all files exist, False otherwise.

run_data_clean_up.**check_args**(*args*)
Checks the arguments and options.

Parameters **args** (argparse.Namespace) – an object containing the options and arguments of the program.

Returns True if everything was OK.

If there is a problem with an option, an exception is raised using the ProgramError class, a message is printed to the sys.stderr and the program exits with error code 1.

run_data_clean_up.**check_input_files**(*prefix*, *the_type*, *required_type*)
Check that the file is of a certain file type.

Parameters

- **prefix** (*string*) – the prefix of the input files.
- **the_type** (*string*) – the type of the input files (bfile, tfile or file).
- **required_type** (*string*) – the required type of the input files (bfile, tfile or file).

Returns True if everything is OK.

Checks if the files are of the required type, according to their current type. The available types are bfile (binary), tfile (transposed) and file (normal).

run_data_clean_up.**main**()
The main function.

These are the steps performed for the data clean up:

1. Prints the version number.
2. Reads the configuration file (`read_config_file()`).
3. Creates a new directory with `data_clean_up` as prefix and the date and time as suffix. In the improbable event that the directory already exists, asks the user the permission to overwrite it.
4. Check the input file type (`bfile`, `tfile` or `file`).
5. Creates an intermediate directory with the section as prefix and the script name as suffix (inside the previous directory).
6. Runs the required script in order (according to the configuration file section).

Note: The main function is not responsible to check if the required files exist. This should be done in the `run` functions.

`run_data_clean_up.`**`parse_args`**`()`
Parses the command line options and arguments.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--bfile`	String	The input binary file prefix from Plink.
`--tfile`	String	The input transposed file prefix from Plink.
`--file`	String	The input file prefix from Plink.
`--conf`	String	The parameter file for the data clean up.
`--overwrite`	Boolean	Overwrites output directories without asking the user.

Note: No option check is done here (except for the one automatically done by `argparse`). Those need to be done elsewhere (see `checkArgs()`).

`run_data_clean_up.`**`read_config_file`**(*filename*)
Reads the configuration file.

Parameters **filename** (*string*) – the name of the file containing the configuration.

Returns A tuple where the first element is a list of sections, and the second element is a map containing the configuration (options and values).

The structure of the configuration file is important. Here is an example of a configuration file:

```
[1] # Computes statistics on duplicated samples
script = duplicated_samples

[2] # Removes samples according to missingness
script = sample_missingness

[3] # Removes markers according to missingness
script = snp_missingness

[4] # Removes samples according to missingness (98%)
script = sample_missingness
mind = 0.02

[5] # Performs a sex check
script = sex_check

[6] # Flags markers with MAF=0
```

```
script = flag_maf_zero

[7] # Flags markers according to Hardy Weinberg
script = flag_hw

[8] # Subset the dataset (excludes markers and remove samples)
script = subset
exclude = .../filename
rempove = .../filename
```

Sections are in square brackets and must be `integer`. The section number represent the step at which the script will be run (*i.e.* from the smallest number to the biggest). The sections must be continous.

Each section contains the script names (`script` variable) and options of the script (all other variables) (*e.g.* section 4 runs the `sample_missingness` script (`run_sample_missingness()`) with option `mind` sets to 0.02).

Here is a list of the available scripts:

- `duplicated_samples` (`run_duplicated_samples()`)
- `duplicated_snps` (`run_duplicated_snps()`)
- `noCall_hetero_snps` (`run_noCall_hetero_snps()`)
- `sample_missingness` (`run_sample_missingness()`)
- `snp_missingness` (`run_snp_missingness()`)
- `sex_check` (`run_sex_check()`)
- `plate_bias` (`run_plate_bias()`)
- `remove_heterozygous_haploid` (`run_remove_heterozygous_haploid()`)
- `find_related_samples` (`run_find_related_samples()`)
- `check_ethnicity` (`run_check_ethnicity()`)
- `flag_maf_zero` (`run_flag_maf_zero()`)
- `flag_hw` (`run_flag_hw()`)
- `subset` (`run_subset_data()`)
- `compare_gold_standard` (`run_compare_gold_standard()`)

`run_data_clean_up.`**`run_check_ethnicity`**(*in_prefix*, *in_type*, *out_prefix*, *options*)

Runs step10 (check ethnicity).

Parameters

- **in_prefix** (*string*) – the prefix of the input files.
- **in_type** (*string*) – the type of the input files.
- **out_prefix** (*string*) – the output prefix.
- **options** (*list of strings*) – the options needed.

Returns a tuple containing the prefix of the output files (the input prefix for the next script) and the type of the output files (`bfile`).

This function calls the `Step10.check_ethnicity` module. The required file type for this module is `bfile`, hence the need to use the `check_input_files()` to check if the file input file type is the good one, or to create it if needed.

Note: The `Step10.check_ethnicity` module doesn't return usable output files. Hence, this function returns the input file prefix and its type.

`run_data_clean_up.`**`run_command`**(*command*)
Run a command using subprocesses.

Parameters **command** (*list of strings*) – the command to run.

Tries to run a command. If it fails, raise a `ProgramError`.

Warning: The variable `command` should be a list of strings (no other type).

`run_data_clean_up.`**`run_compare_gold_standard`**(*in_prefix*, *in_type*, *out_prefix*, *options*)
Compares with a gold standard data set (compare_gold_standard.

Parameters

- **in_prefix** (*string*) – the prefix of the input files.
- **in_type** (*string*) – the type of the input files.
- **out_prefix** (*string*) – the output prefix.
- **options** (*list of strings*) – the options needed.

Returns a tuple containing the prefix of the output files (the input prefix for the next script) and the type of the output files (`bfile`).

This function calls the `Misc.compare_gold_standard` module. The required file type for this module is `bfile`, hence the need to use the `check_input_files()` to check if the file input file type is the good one, or to create it if needed.

Note: The `Misc.compare_gold_standard` module doesn't return usable output files. Hence, this function returns the input file prefix and its type.

`run_data_clean_up.`**`run_duplicated_samples`**(*in_prefix*, *in_type*, *out_prefix*, *options*)
Runs step1 (duplicated samples).

Parameters

- **in_prefix** (*string*) – the prefix of the input files.
- **in_type** (*string*) – the type of the input files.
- **out_prefix** (*string*) – the output prefix.
- **options** (*list of strings*) – the options needed.

Returns a tuple containing the prefix of the output files (the input prefix for the next script) and the type of the output files (`tfile`).

This function calls the `Step1.duplicated_samples` module. The required file type for this module is `tfile`, hence the need to use the `check_input_files()` to check if the file input file type is the good one, or to create it if needed.

`run_data_clean_up.`**`run_duplicated_snps`**(*in_prefix*, *in_type*, *out_prefix*, *options*)
Runs step2 (duplicated snps).

Parameters

- **in_prefix** (*string*) – the prefix of the input files.

- **in_type** (*string*) – the type of the input files.
- **out_prefix** (*string*) – the output prefix.
- **options** (*list of strings*) – the options needed.

Returns a tuple containing the prefix of the output files (the input prefix for the next script) and the type of the output files (`tfile`).

This function calls the `Step2.duplicated_snps` module. The required file type for this module is `tfile`, hence the need to use the `check_input_files()` to check if the file input file type is the good one, or to create it if needed.

Note: This function creates a `map` file, needed for the `Step2.duplicated_snps` module.

`run_data_clean_up.`**`run_find_related_samples`**(*in_prefix*, *in_type*, *out_prefix*, *options*)
Runs step9 (find related samples).

Parameters

- **in_prefix** (*string*) – the prefix of the input files.
- **in_type** (*string*) – the type of the input files.
- **out_prefix** (*string*) – the output prefix.
- **options** (*list of strings*) – the options needed.

Returns a tuple containing the prefix of the output files (the input prefix for the next script) and the type of the output files (`bfile`).

This function calls the `Step9.find_related_samples` module. The required file type for this module is `bfile`, hence the need to use the `check_input_files()` to check if the file input file type is the good one, or to create it if needed.

Note: The `Step9.find_related_samples` module doesn't return usable output files. Hence, this function returns the input file prefix and its type.

`run_data_clean_up.`**`run_flag_hw`**(*in_prefix*, *in_type*, *out_prefix*, *options*)
Runs step12 (flag HW).

Parameters

- **in_prefix** (*string*) – the prefix of the input files.
- **in_type** (*string*) – the type of the input files.
- **out_prefix** (*string*) – the output prefix.
- **options** (*list of strings*) – the options needed.

Returns a tuple containing the prefix of the output files (the input prefix for the next script) and the type of the output files (`bfile`).

This function calls the `Step12.flag_hw` module. The required file type for this module is `bfile`, hence the need to use the `check_input_files()` to check if the file input file type is the good one, or to create it if needed.

Note: The `Step12.flag_hw` module doesn't return usable output files. Hence, this function returns the input file prefix and its type.

---

run_data_clean_up.**run_flag_maf_zero**(*in_prefix*, *in_type*, *out_prefix*, *options*)
Runs step11 (flag MAF zero).

Parameters

- **in_prefix** (*string*) – the prefix of the input files.
- **in_type** (*string*) – the type of the input files.
- **out_prefix** (*string*) – the output prefix.
- **options** (*list of strings*) – the options needed.

Returns a tuple containing the prefix of the output files (the input prefix for the next script) and the type of the output files (`bfile`).

This function calls the `Step11.flag_maf_zero` module. The required file type for this module is `bfile`, hence the need to use the `check_input_files()` to check if the file input file type is the good one, or to create it if needed.

Note: The `Step11.flag_maf_zero` module doesn't return usable output files. Hence, this function returns the input file prefix and its type.

---

run_data_clean_up.**run_noCall_hetero_snps**(*in_prefix*, *in_type*, *out_prefix*, *options*)
Runs step 3 (clean no call and hetero).

Parameters

- **in_prefix** (*string*) – the prefix of the input files.
- **in_type** (*string*) – the type of the input files.
- **out_prefix** (*string*) – the output prefix.
- **options** (*list of strings*) – the options needed.

Returns a tuple containing the prefix of the output files (the input prefix for the next script) and the type of the output files (`tfile`).

This function calls the `Step3.clean_noCall_hetero_snps` module. The required file type for this module is `tfile`, hence the need to use the `check_input_files()` to check if the file input file type is the good one, or to create it if needed.

run_data_clean_up.**run_plate_bias**(*in_prefix*, *in_type*, *out_prefix*, *options*)
Runs step7 (plate bias).

Parameters

- **in_prefix** (*string*) – the prefix of the input files.
- **in_type** (*string*) – the type of the input files.
- **out_prefix** (*string*) – the output prefix.
- **options** (*list of strings*) – the options needed.

Returns a tuple containing the prefix of the output files (the input prefix for the next script) and the type of the output files (`bfile`).

This function calls the `Step7.plate_bias` module. The required file type for this module is `bfile`, hence the need to use the `check_input_files()` to check if the file input file type is the good one, or to create it if needed.

Note: The `Step7.plate_bias` module doesn't return usable output files. Hence, this function returns the input file prefix and its type.

`run_data_clean_up.`**`run_remove_heterozygous_haploid`**(*in_prefix*, *in_type*, *out_prefix*, *options*)

Runs step8 (remove heterozygous haploid).

Parameters

- **in_prefix** (*string*) – the prefix of the input files.
- **in_type** (*string*) – the type of the input files.
- **out_prefix** (*string*) – the output prefix.
- **options** (*list of strings*) – the options needed.

Returns a tuple containing the prefix of the output files (the input prefix for the next script) and the type of the output files (`bfile`).

This function calls the `Step8.remove_heterozygous_haploid` module. The required file type for this module is `bfile`, hence the need to use the `check_input_files()` to check if the file input file type is the good one, or to create it if needed.

`run_data_clean_up.`**`run_sample_missingness`**(*in_prefix*, *in_type*, *out_prefix*, *options*)

Runs step4 (clean mind).

Parameters

- **in_prefix** (*string*) – the prefix of the input files.
- **in_type** (*string*) – the type of the input files.
- **out_prefix** (*string*) – the output prefix.
- **options** (*list of strings*) – the options needed.

Returns a tuple containing the prefix of the output files (the input prefix for the next script) and the type of the output files (`bfile`).

This function calls the `Step4.sample_missingness` module. The required file type for this module is either a `bfile` or a `tfile`, hence the need to use the `check_input_files()` to check if the file input file type is the good one, or to create it if needed.

`run_data_clean_up.`**`run_sex_check`**(*in_prefix*, *in_type*, *out_prefix*, *options*)

Runs step6 (sexcheck).

Parameters

- **in_prefix** (*string*) – the prefix of the input files.
- **in_type** (*string*) – the type of the input files.
- **out_prefix** (*string*) – the output prefix.
- **options** (*list of strings*) – the options needed.

Returns a tuple containing the prefix of the output files (the input prefix for the next script) and the type of the output files (`bfile`).

This function calls the `Step6.sex_check` module. The required file type for this module is `bfile`, hence the need to use the `check_input_files()` to check if the file input file type is the good one, or to create it if needed.

Note: The `Step6.sex_check` module doesn't return usable output files. Hence, this function returns the input file prefix and its type.

`run_data_clean_up.`**`run_snp_missingness`**(*in_prefix*, *in_type*, *out_prefix*, *options*)
Run step5 (clean geno).

Parameters

- **in_prefix** (*string*) – the prefix of the input files.
- **in_type** (*string*) – the type of the input files.
- **out_prefix** (*string*) – the output prefix.
- **options** (*list of strings*) – the options needed.

Returns a tuple containing the prefix of the output files (the input prefix for the next script) and the type of the output files (`bfile`).

This function calls the `Step5.snp_missingness` module. The required file type for this module is `bfile`, hence the need to use the `check_input_files()` to check if the file input file type is the good one, or to create it if needed.

`run_data_clean_up.`**`run_subset_data`**(*in_prefix*, *in_type*, *out_prefix*, *options*)
Subsets the data.

Parameters

- **in_prefix** (*string*) – the prefix of the input files.
- **in_type** (*string*) – the type of the input files.
- **out_prefix** (*string*) – the output prefix.
- **options** (*list of strings*) – the options needed.

Returns a tuple containing the prefix of the output files (the input prefix for the next script) and the type of the output files (`bfile`).

This function calls the `PlinkUtils.subset_data` module. The required file type for this module is `bfile`, hence the need to use the `check_input_files()` to check if the file input file type is the good one, or to create it if needed.

Note: The output file type is the same as the input file type.

5.2 Duplicated Samples Module

The usage of the standalone module is shown below:

```
$ pyGenClean_duplicated_samples --help
usage: pyGenClean_duplicated_samples [-h] --tfile FILE
                                     [--sample-completion-threshold FLOAT]
                                     [--sample-concordance-threshold FLOAT]
                                     [--out FILE]

Extract and work with duplicated samples.

optional arguments:
  -h, --help            show this help message and exit
```

```
Input File:
  --tfile FILE          The input file prefix (will find the tped and tfam
                        file by appending the prefix to .tped and .tfam,
                        respectively.) The duplicated samples should have the
                        same identification numbers (both family and
                        individual ids.)

Options:
  --sample-completion-threshold FLOAT
                        The completion threshold to consider a replicate when
                        choosing the best replicates and for creating the
                        composite samples. [default: 0.9]
  --sample-concordance-threshold FLOAT
                        The concordance threshold to consider a replicate when
                        choosing the best replicates and for creating the
                        composite samples. [default: 0.97]

Output File:
  --out FILE            The prefix of the output files. [default: dup_samples]
```

5.2.1 Input Files

This module uses PLINK's transposed pedfile format (`tped` and `tfam` files). For this step to work, the duplicated samples must have the same identification (family and sample ID). One should keep a file containing the original identifications before modifying the dataset accordingly.

5.2.2 Procedure

Here are the steps performed by the module:

1. Reads the `tfam` file to find duplicated samples.
2. Separates the duplicated samples from the unique samples.
3. Writes the unique samples into a file.
4. Reads the `tped` file and write the pedigree file for the unique samples. Saves in memory the pedigree for the duplicated samples. Updates the indexes of the duplicated samples.
5. If there are no duplicated samples, simply create the final file. Stop here.
6. Computes the completion (for each of the duplicated samples) and the concordance of each sample pairs.
7. Prints statistics (concordance and completion).
8. Prints the concordance matrix for each duplicated samples.
9. Prints the `tped` and the `tfam` file for the duplicated samples.
10. Chooses the best of each duplicates (to keep and to complete) according to completion and concordance.
11. Creates a unique `tped` and `tfam` from the duplicated samples by completing the best chosen one with the other samples.
12. Creates the final dataset.

5.2.3 Output Files

The output files of each of the steps described above are as follow (note that the output prefix shown is the one by default [*i.e.* dup_samples]):

1. No output file is created.
2. No output file is created.
3. Only one of the two PLINK's transposed pedfiles is created:
   - `dup_samples.unique_samples.tfam`: the `tfam` file containing only the unique samples from the original dataset.
4. The second of the two PLINK's transposed pedfiles is created (see previous step):
   - `dup_samples.unique_samples.tped`: the `tped` file containing only the unique samples from the original dataset.
5. If there are not duplicated samples, the final PLINK's transposed pedfiles are created (if not, continue tu next step):
   - `dup_samples.final`: the `tfam` and `tped` final files.
6. One result file is created:
   - `dup_samples.diff`: a file containing the differences in the genotypes for each pair of duplicated samples. Each line contains the following information: `name` the name of the marker, `famID` the family ID, `indID` the individual ID, `dupIndex_1` the index of the first duplicated sample in the original dataset (since the identification of each duplicated samples are the same), `dupIndex_2` the index of the second duplicated sample in the original dataset, `genotype_1` and `genotype_2`, the genotype of the first and second duplicated samples for the current marker, respectively.
7. One result file is created:
   - `dup_samples.summary`: the completion and summarized concordance of each duplicated sample pair. The first two columns (`origIndex` and `dupIndex` are the indexes of the duplicated sample in the original and duplicated transposed pedfiles, respectively.
8. One result file is created
   - `dup_samples.concordance`: the pairwise concordance of each duplicated samples.
9. One set of PLINK's transposed pedfiles:
   - `dup_samples.duplicated_samples`: the dataset containing the duplicated samples from the original dataset.
10. Two output files are created:
    - `dup_samples.chosen_samples.info`: a list of samples that were chosen for completion according to their completion and summarized concordance with their duplicates. Again, their indexes in the original and duplicated transposed pedfiles are saved (the two first columns).
    - `dup_samples.excluded_samples.info`: a list of samples that were not chosen for completion according to their completion and summarized concordance with their duplicates.
11. Multiple output files are created, along with on set of PLINK's transposed pedfiles:
    - `dup_samples.zeroed_out`: the list of genotypes that were zeroed out during completion of the chosen duplicated samples.
    - `dup_samples.not_good_enough`: the list of samples that were not good enough (according to completion and concordance) to create the composite sample (the chosen duplicated samples).

12. Two sets of PLINK's transposed pedfiles are created:
    - `dup_samples.chosen_samples`: a transposed pedfiles containing the completed chosen samples.
    - `dup_samples.final`: the final dataset.

5.2.4 The Algorithm

For more information about the actual algorithms and source codes (the `Step1.duplicated_samples` module), refer to the following sections.

Step1.duplicated_samples

exception `Step1.duplicated_samples.`**`ProgramError`**(*msg*)
An `Exception` raised in case of a problem.

Parameters **msg** (*string*) – the message to print to the user before exiting.

`Step1.duplicated_samples.`**`addToTPEDandTFAM`**(*tped*, *tfam*, *prefix*, *toAddPrefix*)
Append a tfile to another, creating a new one.

Parameters

- **tped** (`numpy.array`) – the `tped` that will be appended to the other one.
- **tfam** (`numpy.array`) – the `tfam` that will be appended to the other one.
- **prefix** (*string*) – the prefix of all the files.
- **toAddPrefix** (*string*) – the prefix of the final file.

Here are the steps of this function:

1. Writes the `tped` into `prefix.chosen_samples.tped`.
2. Writes the `tfam` into `prefix.chosen_samples.tfam`.
3. Copies the previous `tfam` (`toAddPrefix.tfam`) into the final `tfam` (`prefix.final.tfam`).
4. Append the `tfam` to the final `tfam` (`prefix.final.tfam`).
5. Reads the previous `tped` (`toAddPrefix.tped`) and append the new `tped` to it, writing the final one (`prefix.final.tped`).

> **Warning:** The `tped` and `tfam` variables need to contain at least one sample.

`Step1.duplicated_samples.`**`checkArgs`**(*args*)
Checks the arguments and options.

Parameters **args** (`argparse.Namespace`) – a `argparse.Namespace` object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the `ProgramError` class, a message is printed to the `sys.stderr` and the program exists with code 1.

`Step1.duplicated_samples.`**`chooseBestDuplicates`**(*tped*, *samples*, *oldSamples*, *completion*, *concordance_all*, *prefix*)
Choose the best duplicates according to the completion rate.

Parameters

- **tped** (`numpy.array`) – the `tped` containing the duplicated samples.
- **samples** (*dict*) – the updated position of the samples in the tped containing only duplicated samples.
- **oldSamples** (*dict*) – the original duplicated sample positions.
- **completion** (`numpy.array`) – the completion of each of the duplicated samples.
- **concordance_all** (*dict*) – the concordance of every duplicated samples.
- **prefix** (*string*) – the prefix of all the files.

Returns a tuple where the first element is a list of the chosen samples' indexes, the second on is the completion and the last one is the concordance (a map).

These are the steps to find the best duplicated sample:

1. Sort the list of concordances.
2. Sort the list of completions.
3. Choose the best of the concordance and put in a set.
4. Choose the best of the completion and put it in a set.
5. Compute the intersection of the two sets. If there is one sample or more, then randomly choose one sample.
6. If the intersection doesn't contain at least one sample, redo steps 3 and 4, but increase the number of chosen best by one. Redo step 5 and 6 (if required).

The chosen samples are written in `prefix.chosen_samples.info`. The rest are written in `prefix.excluded_samples.info`.

`Step1.duplicated_samples.`**`computeStatistics`**(*tped*, *tfam*, *samples*, *oldSamples*, *prefix*)

Computes the completion and concordance of each samples.

Parameters

- **tped** (`numpy.array`) – the `tped` containing duplicated samples.
- **tfam** (`numpy.array`) – the `tfam` containing duplicated samples.
- **samples** (*dict*) – the updated position of the samples in the tped containing only duplicated samples.
- **oldSamples** (*dict*) – the original duplicated sample positions.
- **prefix** (*string*) – the prefix of all the files.

Returns a tuple containing the completion (`numpy.array`) as first element, and the concordance (`dict`) as last element.

Reads the `tped` file and compute the completion for each duplicated samples and the pairwise concordance between duplicated samples.

Note: The completion and concordance computation excludes a markers if it's on chromosome 24 and if the sample is a female.

Note: A missing genotype is encoded by `0`.

Note: No percentage is computed here, only the numbers. Percentages are computing in other functions: `printStatistics()`, for completion, and `printConcordance()`, for concordance.

Completion

Computes the completion of none zero values (where all genotypes of at least one duplicated sample are no call [*i.e.* `0`]). The completion of sample i (*i.e.* $Completion_i$) is the number of genotypes that have a call divided by the total number of genotypes (the set G_i):

$$Completion_i = \frac{||g \in G_i \text{ where } g \neq 0||}{||G_i||}$$

Note: We consider a genotype as being missing if the sample is a male and if a marker on chromosome 23 or 24 is heterozygous.

Concordance

Computes the pairwise concordance between duplicated samples. For each marker, if both genotypes are not missing, we add one to the total number of compared markers. If both genotypes are the same, we add one to the number of concordant calls. We write the observed genotype difference in the file `prefix.diff`. The concordance between sample i and j (*i.e.* $Concordance_{i,j}$) is the number of genotypes that are equal divided by the total number of genotypes (excluding the no calls):

$$Concordance_{i,j} = \frac{||g \in G_i \cup G_j \text{ where } g_i = g_j \neq 0||}{||g \in G_i \cup G_j \text{ where } g \neq 0||}$$

`Step1.duplicated_samples.`**`createAndCleanTPED`**(*tped*, *tfam*, *samples*, *oldSamples*, *chosenSamples*, *prefix*, *completion*, *completionT*, *concordance*, *concordanceT*)

Complete a TPED for duplicate samples.

Parameters

- **tped** (`numpy.array`) – the `tped` containing the duplicated samples.
- **tfam** (`numpy.array`) – the `tfam` containing the duplicated samples.
- **samples** (*dict*) – the updated position of the samples in the `tped` containing only duplicated samples.
- **oldSamples** (*dict*) – the original duplicated sample positions.
- **chosenSamples** (*dict*) – the position of the chosen samples.
- **prefix** (*string*) – the prefix of all the files.
- **completion** (`numpy.array`) – the completion of each of the duplicated samples.
- **completionT** (*float*) – the completion threshold.
- **concordance** (*dict*) – the pairwise concordance of each of the duplicated samples.
- **concordanceT** (*float*) – the concordance threshold.

Using a `tped` containing duplicated samples, it creates a `tped` containing unique samples by completing a chosen sample with the other replicates.

Note: A chosen sample is not completed using bad replicates (those that don't have a concordance or a completion higher than a certain threshold). The bad replicates are written in the file `prefix.not_good_enough`.

`Step1.duplicated_samples.`**`findDuplicates`**(*tfam*)
Finds the duplicates in a TFAM.

Parameters **tfam** (*list of tuple of string*) – representation of a `tfam` file.

Returns two `dict`, containing unique and duplicated samples position.

`Step1.duplicated_samples.`**`main`**(*argString=None*)
Check for duplicated samples in a tfam/tped file.

Parameters **argString** (*list of strings*) – the options

Here are the steps for the duplicated samples step.

1. Prints the options.
2. Reads the `tfam` file (`readTFAM()`).
3. Separate the duplicated samples from the unique samples (`findDuplicates()`).
4. Writes the unique samples into a file named `prefix.unique_samples.tfam` (`printUniqueTFAM()`).
5. Reads the `tped` file and write into `prefix.unique_samples.tped` the pedigree file for the unique samples (`processTPED()`). Saves in memory the pedigree for the duplicated samples. Updates the indexes of the duplicated samples.
6. If there are no duplicated samples, simply copies the files `prefix.unique_samples` (`tped` and `tfam`) to `prefix.final.tfam` and `prefix..final.tped`, respectively.
7. Computes the completion (for each of the duplicated samples) and the concordance of each sample pairs (`computeStatistics()`).
8. Prints statistics (concordance and completion) (`printStatistics()`).
9. We print the concordance matrix for each duplicated samples (`printConcordance()`).
10. We print the `tped` and the `tfam` file for the duplicated samples (`prefix.duplicated_samples`) (`printDuplicatedTPEDandTFAM()`).
11. Choose the best of each duplicates (to keep and to complete) according to completion and concordance (`chooseBestDuplicates()`).
12. Creates a unique `tped` and `tfam` from the duplicated samples by completing the best chosen one with the other samples (`createAndCleanTPED()`).
13. Merge the two tfiles together (`prefix.unique_samples` and `prefix.chosen_samples`) to create the final dataset (`prefix.final`) (`addToTPEDandTFAM()`).

`Step1.duplicated_samples.`**`parseArgs`**(*argString=None*)
Parses the command line options and arguments.

Parameters **argString** (*list of strings*) – the options

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--tfile`	string	The input file prefix (of type `tfile`).
`--sample-completion-threshold`	float	The completion threshold.
`--sample-concordance-threshold`	float	The concordance threshold.
`--out`	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see `checkArgs()`).

`Step1.duplicated_samples.`**`printConcordance`**(*concordance*, *prefix*)
Print the concordance.

Parameters

- **concordance** (*dict*) – the concordance of each sample.
- **prefix** (*string*) – the prefix of all the files.

Returns the concordance percentage (dict)

The concordance is the number of genotypes that are equal when comparing a duplicated samples with another one, divided by the total number of genotypes (excluding genotypes that are no call [*i.e.* `0`]). If a duplicated sample has 100% of no calls, the concordance will be zero.

The file `prefix.concordance` will contain $N \times N$ matrices for each set of duplicated samples.

`Step1.duplicated_samples.`**`printDuplicatedTPEDandTFAM`**(*tped*, *tfam*, *samples*, *oldSamples*, *prefix*)
Print the TPED and TFAM of the duplicated samples.

Parameters

- **tped** (`numpy.array`) – the `tped` containing duplicated samples.
- **tfam** (`numpy.array`) – the `tfam` containing duplicated samples.
- **samples** (*dict*) – the updated position of the samples in the tped containing only duplicated samples.
- **oldSamples** (*dict*) – the original duplicated sample positions.
- **prefix** (*string*) – the prefix of all the files.

The `tped` and `tfam` files are written in `prefix.duplicated_samples.tped` and `prefix.duplicated_samples.tfam`, respectively.

`Step1.duplicated_samples.`**`printStatistics`**(*completion*, *concordance*, *tpedSamples*, *oldSamples*, *prefix*)
Print the statistics in a file.

Parameters

- **completion** (`numpy.array`) – the completion of each duplicated samples.
- **concordance** (*dict*) – the concordance of each duplicated samples.
- **tpedSamples** (*dict*) – the updated position of the samples in the tped containing only duplicated samples.
- **oldSamples** (*dict*) – the original duplicated sample positions.
- **prefix** (*string*) – the prefix of all the files.

Returns the completion for each duplicated samples, as a `numpy.array`.

Prints the statistics (completion of each samples and pairwise concordance between duplicated samples) in a file (`prefix.summary`).

`Step1.duplicated_samples.`**`printUniqueTFAM`**(*tfam*, *samples*, *prefix*)
Prints a new TFAM with only unique samples.

Parameters

- **tfam** (*list of tuples of strings*) – a representation of a TFAM file.
- **samples** (*dict*) – the position of the samples
- **prefix** (*string*) – the prefix of the output file name

`Step1.duplicated_samples.`**`processTPED`**(*uniqueSamples*, *duplicatedSamples*, *fileName*, *prefix*)
Process the TPED file.

Parameters

- **uniqueSamples** (*dict*) – the position of unique samples.
- **duplicatedSamples** (*collections.defaultdict*) – the position of duplicated samples.
- **fileName** (*string*) – the name of the file.
- **prefix** (*string*) – the prefix of all the files.

Returns a tuple containing the `tped` (`numpy.array`) as first element, and the updated positions of the duplicated samples (`dict`)

Reads the entire `tped` and prints another one containing only unique samples (`prefix.unique_samples.tped`). It then creates a `numpy.array` containing the duplicated samples.

`Step1.duplicated_samples.`**`readTFAM`**(*fileName*)
Reads the TFAM file.

Parameters **fileName** (*string*) – the name of the `tfam` file.

Returns a list of tuples, representing the `tfam` file.

5.3 Duplicated Markers Module

The usage of the standalone module is shown below:

```
$ pyGenClean_duplicated_snps --help
usage: pyGenClean_duplicated_snps [-h] --tfile FILE
                                  [--snp-completion-threshold FLOAT]
                                  [--snp-concordance-threshold FLOAT]
                                  [--frequency_difference FLOAT] [--out FILE]

Extract and work with duplicated SNPs.

optional arguments:
  -h, --help            show this help message and exit

Input File:
  --tfile FILE          The input file prefix (will find the tped and tfam
                        file by appending the prefix to .tped and .tfam,
                        respectively. A .map file is also required.

Options:
  --snp-completion-threshold FLOAT
                        The completion threshold to consider a replicate when
                        choosing the best replicates and for composite
                        creation. [default: 0.9]
  --snp-concordance-threshold FLOAT
```

```
                        The concordance threshold to consider a replicate when
                        choosing the best replicates and for composite
                        creation. [default: 0.98]
  --frequency_difference FLOAT
                        The maximum difference in frequency between duplicated
                        markers [default: 0.05]

Output File:
  --out FILE            The prefix of the output files. [default: dup_snps]
```

5.3.1 Input Files

This module uses PLINK's transposed pedfile format (`tped` and `tfam` files). It also requires a `map` file to speed up the process of finding the duplicated markers, so that the `tped` file is not read.

5.3.2 Procedure

Here are the steps performed by the module:

1. Reads the `map` file to gather marker's position.
2. Reads the `tfam` file.
3. Finds the unique markers using the `map` file.
4. Process the `tped` file, finding unique and duplicated markers according to chromosomal positions.
5. If there are no duplicated markers, stop here.
6. If there are duplicated markers, print a `tped` and `tfam` file containing the duplicated markers.
7. Computes the frequency of the duplicated markers (using Plink) and read the output file.
8. Computes the concordance and pairwise completion of each of the duplicated markers.
9. Prints the problematic duplicated markers with a file containing the summary of the statistics (completion and pairwise concordance).
10. Print the pairwise concordance in a file (matrices).
11. Choose the best duplicated markers using concordance and completion.
12. Completes the chosen markers with the remaining duplicated markers.
13. Creates the final `tped` file, containing the unique markers, the chosen duplicated markers that were completed, and the problematic duplicated markers (for further analysis). This set excludes markers that were used for completing the chosen ones.

5.3.3 Output Files

The output files of each of the steps described above are as follow (note that the output prefix shown is the one by default [*i.e.* dup_snps]):

1. If the marker names are not unique, one file is created:
   - `dup_snps.duplicated_marker_names`: a list of marker names and chromosomal positions for each marker with duplicated names. This file is not created if there are no markers with duplicated names.
2. No files are created.

3. No files are created.
4. One set of transposed pedfiles.
    - `dup_snps.unique_snps`: the transposed pedfiles containing the unique markers (according to chromosomal positions).
5. If there are no duplicated markers (according to chromosomal positions), the transposed pedfiles created at the previous step are copied to a new set of transposed pedfiles.
    - `dup_snps.final`: the final transposed pedfiles.
6. One set of transposed pedfiles.
    - `dup_snps.duplicated_snps`: the transposed pedfiles containing the duplicated markers (according to chromosomal positions).
7. One set of PLINK's result file.
    - `dup_snps.duplicated_snps`: the file with the `frq` extension contains the frequency of each duplicated markers.
8. No files are created.
9. Multiple files are created.
    - `dup_snps.summary`: contains the completion and pairwise concordance between duplicated markers.
    - `dup_snps.problems`: contains the list of markers with "problems" that can't be used for further completion of the duplicated markers. (either a difference in MAF [`diff_frequency`], a difference in the minor allele [`diff_minor_allele`], two homozygous markers where one is flipped [`homo_flip`], markers with flipped alleles [`flip`], one marker is homozygous, the other is heterozygous [`homo_hetero`], one marker is homozygous, the other is heterozygous but one is flipped [`homo_hetero_flip`] or any other problem [`problem`].
10. One output file is created.
    - `dup_snps.concordance`: a matrix containing a pairwise concordance comparison for each duplicated markers.
11. Two output files are created.
    - `dup_snps.chosen_snps.info`: the list of duplicated markers that were chosen for completion with the other markers (the best of the duplicated markers, according to concordance and completion).
    - `dup_snps.not_chosen_snps.info`: the list of duplicated markers that were not chosen for completion with the other markers.
12. Multiple output files are created along with a set of transposed pedfiles.
    - `dup_snps.zeroed_out`: the list of genotypes that were zeroed out while completing the chosen duplicated markers with the others. Each line contains the id of the sample and the name of the marker that was zeroed out.
    - `dup_snps.not_good_enough`: the list of markers that were not good enough (according to concordance and completion) to complete the best of the duplicated markers.
    - `dup_snps.removed_duplicates`: the list of markers that were used to complete the chosen duplicated markers. Those markers were removed from the dataset.
    - `dup_snps.chosen_snps`: the transposed pedfiles containing the completed chosen duplicated markers (a composite of all the duplicated markers that were good enough).
13. On set of transposed pedfiles.

- `dup_snps.final`: the final dataset, containing the unique markers, the chosen duplicated markers that were complete (composite) and the duplicated markers that weren't completed because of various problems.

5.3.4 The Algorithm

For more information about the actual algorithms and source codes (the `Step2.duplicated_snps` module), refer to the following sections.

Step2.duplicated_snps

exception `Step2.duplicated_snps.`**`ProgramError`**(*msg*)
An `Exception` raised in case of a problem.

Parameters **msg** (*string*) – the message to print to the user before exiting.

`Step2.duplicated_snps.`**`checkArgs`**(*args*)
Checks the arguments and options.

Parameters **args** (*argparse.Namespace*) – an object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the `ProgramError` class, a message is printed to the `sys.stderr` and the program exists with code 1.

`Step2.duplicated_snps.`**`chooseBestSnps`**(*tped*, *snps*, *trueCompletion*, *trueConcordance*, *prefix*)
Choose the best duplicates according to the completion and concordance.

Parameters

- **tped** (*numpy.array*) – a representation of the `tped` of duplicated markers.
- **snps** (*dict*) – the position of the duplicated markers in the `tped`.
- **trueCompletion** (*numpy.array*) – the completion of each markers.
- **trueConcordance** (*dict*) – the pairwise concordance of each markers.
- **prefix** (*string*) – the prefix of the output files.

Returns a tuple containing the chosen indexes (`dict`) as the first element, the completion (`numpy.array`) as the second element, and the concordance (`dict`) as last element.

It creates two output files: `prefix.chosen_snps.info` and `prefix.not_chosen_snps.info`. The first one contains the markers that were chosen for completion, and the second one, the markers that weren't.

It starts by computing the completion of each markers (dividing the number of calls divided by the total number of genotypes). Then, for each of the duplicated markers, we choose the best one according to completion and concordance (see explanation in `Step1.duplicated_samples.chooseBestDuplicates()` for more details).

`Step2.duplicated_snps.`**`computeFrequency`**(*prefix*, *outPrefix*)
Computes the frequency of the SNPs using Plink.

Parameters

- **prefix** (*string*) – the prefix of the input files.
- **outPrefix** (*string*) – the prefix of the output files.

Returns a `dict` containing the frequency of each marker.

Start by computing the frequency of all markers using Plink. Then, it reads the output file, and saves the frequency and allele information.

`Step2.duplicated_snps.`**`computeStatistics`**(*tped*, *tfam*, *snps*)

Computes the completion and concordance of each SNPs.

Parameters

- **tped** (*numpy.array*) – a representation of the `tped`.
- **tfam** (*list of tuples of strings*) – a representation of the `tfam`
- **snps** (*dict*) – the position of the duplicated markers in the `tped`.

Returns a tuple containing the completion of duplicated markers (`numpy.array`) as first element, and the concordance (`dict`) of duplicated markers, as last element.

A marker's completion is compute using this formula (where G_i is the set of genotypes for the marker i):

$$Completion_i = \frac{||g \in G_i \text{ where } g \neq 0||}{||G_i||}$$

The pairwise concordance between duplicated markers is compute as follow (where G_i and G_j are the sets of genotypes for markers i and j, respectively):

$$Concordance_{i,j} = \frac{||g \in G_i \cup G_j \text{ where } g_i = g_j \neq 0||}{||g \in G_i \cup G_j \text{ where } g \neq 0||}$$

Hence, we only computes the numerators and denominators of the completion and concordance, for future reference.

Note: When the genotypes are not comparable, the function tries to flip one of the genotype to see if it becomes comparable.

`Step2.duplicated_snps.`**`createAndCleanTPED`**(*tped*, *tfam*, *snps*, *prefix*, *chosenSNPs*, *completion*, *concordance*, *snpsToComplete*, *tfamFileName*, *completionT*, *concordanceT*)

Complete a TPED for duplicated SNPs.

Parameters

- **tped** (*numpy.array*) – a representation of the `tped` of duplicated markers.
- **tfam** (*list of tuples of strings*) – a representation of the `tfam`.
- **snps** (*dict*) – the position of duplicated markers in the `tped`.
- **prefix** (*string*) – the prefix of the output files.
- **chosenSNPs** (*dict*) – the markers that were chosen for completion (including problems).
- **completion** (*numpy.array*) – the completion of each of the duplicated markers.
- **concordance** (*dict*) – the pairwise concordance of the duplicated markers.
- **snpsToComplete** (*set*) – the markers that will be completed (excluding problems).
- **tfamFileName** (*string*) – the name of the original `tfam` file.
- **completionT** (*float*) – the completion threshold.
- **concordanceT** (*float*) – the concordance threshold.

Returns a tuple containing the new `tped` after completion (`numpy.array` as the first element, and the index of the markers that will need to be rid of (`set`) as the last element.

It creates three different files:

- **`prefix.zeroed_out`: contains information about markers and samples** where the genotyped was zeroed out.
- **`prefix.not_good_enough`: contains information about markers that were** not good enough to help in completing the chosen markers (because of concordance or completion).
- **`prefix.removed_duplicates`: the list of markers that where used for** completing the chosen one, hence they will be removed from the final data set.

Cycling through every genotypes of every samples of every duplicated markers, checks if the genotypes are all the same. If the chosen one was not called, but the other ones were, then we complete the chosen one with the genotypes for the others (assuming that they are all the same). If there is a difference between the genotypes, it is zeroed out for the chosen marker.

`Step2.duplicated_snps.`**`createFinalTPEDandTFAM`**(*tped*, *toReadPrefix*, *prefix*, *snpToRemove*)
Creates the final TPED and TFAM.

Parameters

- **tped** (*numpy.array*) – a representation of the `tped` of duplicated markers.
- **toReadPrefix** (*string*) – the prefix of the unique files.
- **prefix** (*string*) – the prefix of the output files.
- **snpToRemove** (*set*) – the markers to remove.

Starts by copying the unique markers' `tfam` file to `prefix.final.tfam`. Then, it copies the unique markers' `tped` file, in which the chosen markers will be appended.

The final data set will include the unique markers, the chosen markers which were completed, and the problematic duplicated markers (for further analysis). The markers that were used to complete the chosen ones are not present in the final data set.

`Step2.duplicated_snps.`**`findUniques`**(*mapF*)
Finds the unique markers in a MAP.

Parameters **mapF** (*list of tuple of string*) – representation of a `map` file.

Returns a `dict` containing unique markers (according to their genomic localisation).

`Step2.duplicated_snps.`**`flipGenotype`**(*genotype*)
Flips a genotype.

Parameters **genotype** (*set*) – the genotype to flip.

Returns the new flipped genotype (as a `set`)

```
>>> flipGenotype({"A", "T"})
set(['A', 'T'])
>>> flipGenotype({"C", "T"})
set(['A', 'G'])
>>> flipGenotype({"T", "G"})
set(['A', 'C'])
>>> flipGenotype({"0", "0"})
Traceback (most recent call last):
    ...
ProgramError: 0: unkown allele
>>> flipGenotype({"A", "N"})
Traceback (most recent call last):
```

```
    ...
ProgramError: N: unkown allele
```

Step2.duplicated_snps.**getIndexOfHeteroMen**(*genotypes*, *menIndex*)

Get the indexes of heterozygous men.

Parameters

- **genotypes** (*numpy.array*) – the genotypes of everybody.
- **menIndex** (*numpy.array*) – the indexes of the men (for the genotypes).

Returns a `numpy.array` containing the indexes of the genotypes to remove.

Finds the mean that have a heterozygous genotype for this current marker. Usually used on sexual chromosomes.

Step2.duplicated_snps.**main**(*argString=None*)

The main function of the module..

Here are the steps for duplicated samples:

1. Prints the options.
2. Reads the `map` file to gather marker's position (`readMAP()`).
3. Reads the `tfam` file (`readTFAM()`).
4. Finds the unique markers using the `map` file (`findUniques()`).
5. Process the `tped` file. Write a file containing unique markers in `prefix.unique_snps` (`tfam` and `tped`). Keep in memory information about the duplicated markers (`tped`) (`processTPED()`).
6. If there are no duplicated markers, the file `prefix.unique_snps` (`tped` and `tfam`) are copied to `prefix.final`.
7. If there are duplicated markers, print a `tped` and `tfam` file containing the duplicated markers (`printDuplicatedTPEDandTFAM()`).
8. Computes the frequency of the duplicated markers (using Plink) and read the output file (`computeFrequency()`).
9. Computes the concordance and pairwise completion of each of the duplicated markers (`computeStatistics()`).
10. Prints the problematic duplicated markers with a file containing the summary of the statistics (completion and pairwise concordance) (`printProblems()`).
11. Print the pairwise concordance in a file (matrices) (`printConcordance()`).
12. Choose the best duplicated markers using concordance and completion (`chooseBestSnps()`).
13. Completes the chosen markers with the remaining duplicated markers (`createAndCleanTPED()`).
14. Creates the final `tped` file, containing the unique markers, the chosen duplicated markers that were completed, and the problematic duplicated markers (for further analysis). This set excludes markers that were used for completing the chosen ones (`createFinalTPEDandTFAM()`).

Step2.duplicated_snps.**parseArgs**(*argString=None*)

Parses the command line options and arguments.

Parameters **argString** (*list of strings*) – the options.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--tfile`	string	The input file prefix (Plink tfile).
`--snp-completion-threshold`	float	The completion threshold to consider a replicate when choosing the best replicates.
`--snp-concordance-threshold`	float	The concordance threshold to consider a replicate when choosing the best replicates.
`--frequency_difference`	float	The maximum difference in frequency between duplicated markers.
`--out`	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see `checkArgs()`).

`Step2.duplicated_snps.`**`printConcordance`**(*concordance*, *prefix*, *tped*, *snps*)
Print the concordance.

Parameters

- **concordance** (*dict*) – the concordance.
- **prefix** (*string*) – the prefix if the output files.
- **tped** (*numpy.array*) – a representation of the `tped` of duplicated markers.
- **snps** (*dict*) – the position of the duplicated markers in the `tped`.

Prints the concordance in a file, in the format of a matrix. For each duplicated markers, the first line (starting with the # signs) contains the name of all the markers in the duplicated markers set. Then a $N \times N$ matrix is printed to file (where N is the number of markers in the duplicated marker list), containing the pairwise concordance.

`Step2.duplicated_snps.`**`printDuplicatedTPEDandTFAM`**(*tped*, *tfamFileName*, *outPrefix*)
Print the duplicated SNPs TPED and TFAM.

Parameters

- **tped** (*numpy.array*) – a representation of the `tped` of duplicated markers.
- **tfamFileName** (*string*) – the name of the original `tfam` file.
- **outPrefix** (*string*) – the output prefix.

First, it copies the original `tfam` into `outPrefix.duplicated_snps.tfam`. Then, it prints the `tped` of duplicated markers in `outPrefix.duplicated_snps.tped`.

`Step2.duplicated_snps.`**`printProblems`**(*completion*, *concordance*, *tped*, *snps*, *frequencies*, *prefix*, *diffFreq*)
Print the statistics.

Parameters

- **completion** (*numpy.array*) – the completion of each duplicated markers.
- **concordance** (*dict*) – the pairwise concordance between duplicated markers.
- **tped** (*numpy.array*) – a representation of the `tped` of duplicated markers.
- **snps** (*dict*) – the positions of the duplicated markers in the `tped`
- **frequencies** (*dict*) – the frequency of each of the duplicated markers.
- **prefix** (*string*) – the prefix of the output files.
- **diffFreq** (*float*) – the frequency difference threshold.

Returns a `set` containing duplicated markers to complete.

Creates a summary file (`prefix.summary`) containing information about duplicated markers: chromosome, position, name, alleles, status, completion percentage, completion number and mean concordance.

The frequency and the minor allele are used to be certain that two duplicated markers are exactly the same marker (and not a tri-allelic one, for example).

For each duplicated markers:

1. Constructs the set of available alleles for the first marker.
2. Constructs the set of available alleles for the second marker.
3. If the two sets are different, but the number of alleles is the same, we try to flip one of the marker. If the two sets are the same, but the number of alleles is 1, we set the status to `homo_flip`. If the markers are heterozygous, we set the status to `flip`.
4. If there is a difference in the number of alleles (one is homozygous, the other, heterozygous), and that there is on allele in common, we set the status to `homo_hetero`. If there are no allele in common, we try to flip one. If the new sets have one allele in common, we set the status to `homo_hetero_flip`.
5. If the sets of available alleles are the same (without flip), we check the frequency and the minor alleles. If the minor allele is different, we set the status to `diff_minor_allele`. If the difference in frequencies is higher than a threshold, we set the status to `diff_frequency`.
6. If all of the above fail, we set the status to `problem`.

Problems are written in the `prefix.problems` file, and contains the following columns: chromosome, position, name and status. This file contains all the markers with a status, as explained above.

`Step2.duplicated_snps.`**`processTPED`**(*uniqueSNPs*, *mapF*, *fileName*, *tfam*, *prefix*)
Process the TPED file.

Parameters

- **uniqueSNPs** (*dict*) – the unique markers.
- **mapF** (*list*) – a representation of the `map` file.
- **fileName** (*string*) – the name of the `tped` file.
- **tfam** (*string*) – the name of the `tfam` file.
- **prefix** (*string*) – the prefix of all the files.

Returns a tuple with the representation of the `tped` file (`numpy.array`) as first element, and the updated position of the duplicated markers in the `tped` representation.

Copies the `tfam` file into `prefix.unique_snps.tfam`. While reading the `tped` file, creates a new one (`prefix.unique_snps.tped`) containing only unique markers.

`Step2.duplicated_snps.`**`readMAP`**(*fileName*, *prefix*)
Reads the MAP file.

Parameters **fileName** (*string*) – the name of the `map` file.

Returns a list of tuples, representing the `map` file.

While reading the `map` file, it saves a file (`prefix.duplicated_marker_names`) containing the name of the unique duplicated markers.

`Step2.duplicated_snps.`**`readTFAM`**(*fileName*)
Reads the TFAM file.

Parameters **fileName** (*string*) – the name of the `tfam` file.

Returns a representation the `tfam` file (`numpy.array`).

`Step2.duplicated_snps.`**`runCommand`**(*command*)
Run the command in Plink.

Parameters **command** (*list of strings*) – the command to run.

Tries to run a command using `subprocess`.

5.4 Clean No Call and Only Heterozygous Markers Module

The usage of the standalone module is shown below:

```
$ pyGenClean_clean_noCall_hetero_snps --help
usage: pyGenClean_clean_noCall_hetero_snps [-h] --tfile FILE [--out FILE]

Removes "no calls" only and heterozygous only markers.

optional arguments:
  -h, --help    show this help message and exit

Input File:
  --tfile FILE  The input file prefix (will find the tped and tfam file by
                appending the prefix to .tped and .tfam, respectively.

Output File:
  --out FILE    The prefix of the output files. [default: clean_noCall_hetero]
```

5.4.1 Input FIles

This module uses the transposed pedfile format separated by tabulations (`tped` and `tfam` files) for the source data set (the data of interest).

5.4.2 Procedure

Here are the steps performed by the module:

1. Reads the transposed pedfiles and extract markers which are all heterozygous or all failed from the dataset.

5.4.3 Output Files

The output files of each of the steps described above are as follow (note that the output prefix shown is the one by default [*i.e.* `clean_noCall_hetero_snps`]):

1. One transposed pedfiles and two custom output files are created:
   - `clean_noCall_hetero`: the transposed pedfiles separated by tabulations containing the new dataset, with markers which are all heterozygous or all failed were removed from the initial dataset.
   - `clean_noCall_hetero.allHetero`: the list of markers which were all heterozygous in the initial dataset.
   - `clean_noCall_hetero.allFailed`: the list of markers which were all failed in the initial dataset.

5.4.4 The Plots

A standalone script has been created so that heterozygosity rates can be visualized using histograms or box plots. This script has not yet been included in the automated pipeline, so it needs to be started manually.

```
$ pyGenClean_heterozygosity_plot --help
usage: pyGenClean_heterozygosity_plot [-h] --tfile FILE [--boxplot]
                                      [--format FORMAT] [--bins INT]
                                      [--xlim FLOAT FLOAT] [--ymax FLOAT]
                                      [--out FILE]

Plot the distribution of the heterozygosity ratio.

optional arguments:
  -h, --help            show this help message and exit

Input File:
  --tfile FILE          The prefix of the transposed file

Options:
  --boxplot             Draw a boxplot instead of a histogram.
  --format FORMAT       The output file format (png, ps, pdf, or X11 formats are
                        available). [default: png]
  --bins INT            The number of bins for the histogram. [default: 100]
  --xlim FLOAT FLOAT    The limit of the x axis (floats).
  --ymax FLOAT          The maximal Y value.

Output File:
  --out FILE            The prefix of the output files. [default:
                        heterozygosity]
```

The script produces either a histogram (see the *Heterozygosity rate histogram* figure) or a box plot (see the *Heterozygosity rate box plot* figure) of samples' heterozygosity rates.

5.4.5 The Algorithm

For more information about the actual algorithms and source codes (the `Step3.clean_noCall_hetero_snps` and `Step3.heterozygosity_plot` modules), refer to the following sections.

Step3.clean_noCall_hetero_snps

exception `Step3.clean_noCall_hetero_snps.`**`ProgramError`**(*msg*)
An `Exception` raised in case of a problem.

Parameters **msg** (*string*) – the message to print to the user before exiting.

`Step3.clean_noCall_hetero_snps.`**`checkArgs`**(*args*)
Checks the arguments and options.

Parameters **args** (*argparse.Namespace*) – an object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the `ProgramError` class, a message is printed to the `sys.stderr` and the program exists with code 1.

`Step3.clean_noCall_hetero_snps.`**`main`**(*argString=None*)
The main function of the module.

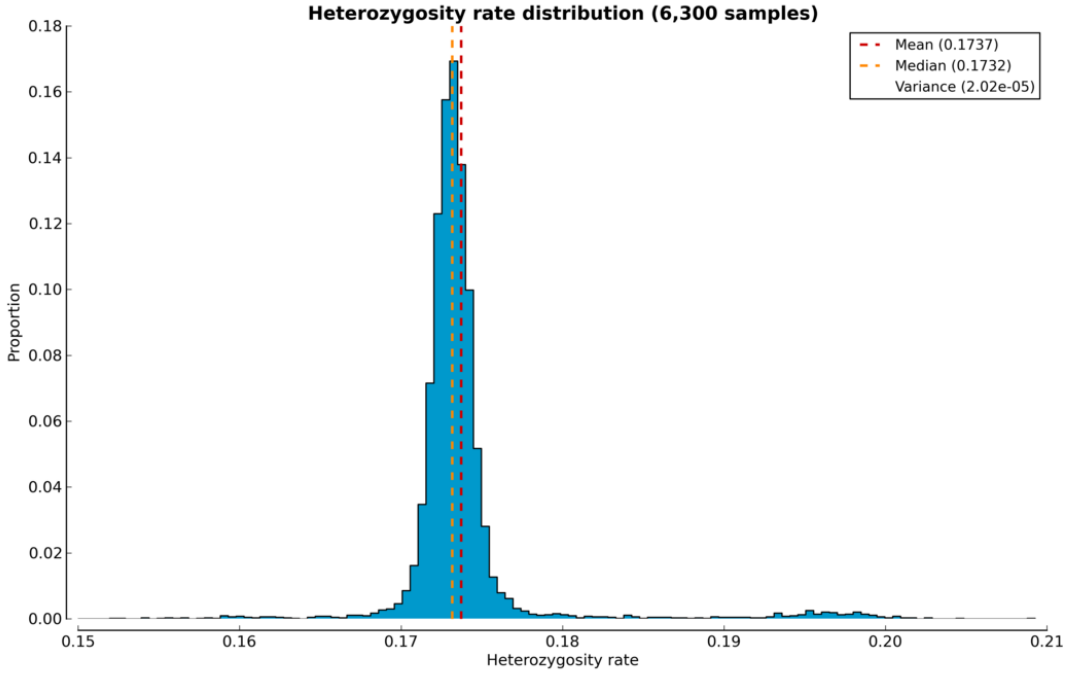


Figure 5.1: Heterozygosity rate histogram

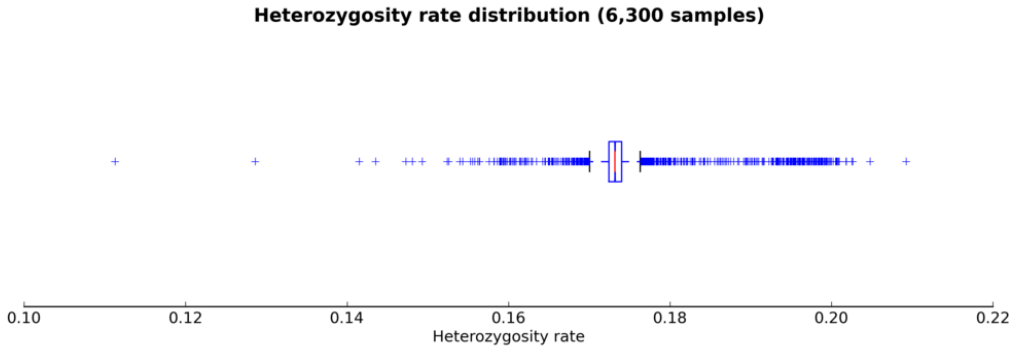


Figure 5.2: Heterozygosity rate box plot

Parameters **argString** (*list of strings*) – the options.

These are the steps:

1. Prints the options.
2. Reads the `tfam` and `tped` files and find all heterozygous and all failed markers (`processTPEDandTFAM()`).

`Step3.clean_noCall_hetero_snps.`**`parseArgs`**(*argString=None*)
Parses the command line options and arguments.

Parameters **argString** (*list of strings*) – the options.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--tfile`	string	The input file prefix (Plink tfile).
`--out`	string	The prefix of the output files

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see `checkArgs()`).

`Step3.clean_noCall_hetero_snps.`**`processTPEDandTFAM`**(*tped*, *tfam*, *prefix*)
Process the TPED and TFAM files.

Parameters

- **tped** (*string*) – the name of the `tped` file.
- **tfam** (*string*) – the name of the `tfam` file.
- **prefix** (*string*) – the prefix of the output files.

Copies the original `tfam` file into `prefix.tfam`. Then, it reads the `tped` file and keeps in memory two sets containing the markers which are all failed or which contains only heterozygous genotypes.

It creates two output files, `prefix.allFailed` and `prefix.allHetero`, containing the markers that are all failed and are all heterozygous, respectively.

Note: All heterozygous markers located on the mitochondrial chromosome are not remove.

Step3.heterozygosity_plot

exception `Step3.heterozygosity_plot.`**`ProgramError`**(*msg*)
An `Exception` raised in case of a problem.

Parameters **msg** (*string*) – the message to print to the user before exiting.

`Step3.heterozygosity_plot.`**`checkArgs`**(*args*)
Checks the arguments and options.

Parameters **args** (*argparse.Namespace*) – an object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the `ProgramError` class, a message is printed to the `sys.stderr` and the program exists with code 1.

Step3.heterozygosity_plot.**compute_heterozygosity**(*in_prefix*, *nb_samples*)
Computes the heterozygosity ratio of samples (from tped).

Step3.heterozygosity_plot.**compute_nb_samples**(*in_prefix*)
Check the number of samples.

Parameters **in_prefix** (*string*) – the prefix of the input file.

Returns the number of sample in `prefix.fam`.

Step3.heterozygosity_plot.**is_heterozygous**(*genotype*)
Tells if a genotype "A B" is heterozygous.

Parameters **genotype** (*string*) – the genotype to test for heterozygosity.

Returns `True` if the genotype is heterozygous, `False` otherwise.

The genotype must contain two alleles, separated by a space. It then compares the first allele (`genotype[0]`) with the last one (`genotype[-1]`).

```
>>> is_heterozygous("A A")
False
>>> is_heterozygous("G C")
True
>>> is_heterozygous("0 0") # No call is not heterozygous.
False
```

Step3.heterozygosity_plot.**main**(*argString=None*)
The main function of the module.

Parameters **argString** (*list of strings*) – the options.

These are the steps:

1. Prints the options.
2. Checks the number of samples in the `tfam` file (`compute_nb_samples()`).
3. Computes the heterozygosity rate (`compute_heterozygosity()`).
4. Plots the heterozygosity rate (`plot_heterozygosity()`).

Step3.heterozygosity_plot.**parseArgs**(*argString=None*)
Parses the command line options and arguments.

Parameters **argString** (*list of strings*) – the options.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--tfile`	string	The prefix of the transposed file.
`--boxplot`	bool	Draw a boxplot instead of a histogram.
`--format`	string	The output file format.
`--bins`	int	The number of bins for the histogram.
`--xlim`	float	The limit of the x axis.
`--ymax`	float	"The maximal Y value.
`--out`	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see `checkArgs()`).

Step3.heterozygosity_plot.**plot_heterozygosity**(*heterozygosity*, *options*)

Plots the heterozygosity rate distribution.

Parameters

- **heterozygosity** (*numpy.array*) –
- **options** (*argparse.Namespace*) – the options.

Plots a histogram or a boxplot of the heterozygosity distribution.

5.5 Sample Missingness Module

The usage of the standalone module is shown below:

```
$ pyGenClean_sample_missingness --help
usage: pyGenClean_sample_missingness [-h] --ifile FILE [--is-bfile]
                                     [--mind FLOAT] [--out FILE]

Computes sample missingness using Plink

optional arguments:
  -h, --help    show this help message and exit

Input File:
  --ifile FILE  The input file prefix (by default, this input file must be a
                tfile. If options --is-bfile is used, the input file must be a
                bfile).

Options:
  --is-bfile    The input file (--ifile) is a bfile instead of a tfile.
  --mind FLOAT  The missingness threshold (remove samples with more than x
                percent missing genotypes). [Default: 0.100]

Output File:
  --out FILE    The prefix of the output files (wich will be a Plink binary
                file). [default: clean_mind]
```

5.5.1 Input FIles

This module uses either PLINK's binary file format (`bed`, `bim` and `fam` files) or the transposed pedfile format separated by tabulations (`tped` and `tfam`) for the source data set (the data of interest).

5.5.2 Procedure

Here are the steps performed by the module:

1. Uses Plink to remove samples with a high missing rate (above a user defined threshold).

5.5.3 Output Files

The output files of each of the steps described above are as follow (note that the output prefix shown is the one by default [*i.e.* `clean_geno`]):

1. One set of PLINK's output and result files:

- `clean_mind`: the new dataset with samples having a high missing rate removed (above a user defined threshold). The file `clean_mind.irem` contains a list of samples that were removed.

5.5.4 The Algorithm

For more information about the actual algorithms and source codes (the `Step4.sample_missingness` module), refer to the following sections.

Step4.sample_missingness

exception `Step4.sample_missingness.`**`ProgramError`**(*msg*)
An `Exception` raised in case of a problem.

Parameters **msg** (*string*) – the message to print to the user before exiting.

`Step4.sample_missingness.`**`checkArgs`**(*args*)
Checks the arguments and options.

Parameters **args** (*argparse.Namespace*) – a object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the `ProgramError` class, a message is printed to the `sys.stderr` and the program exists with code 1.

`Step4.sample_missingness.`**`main`**(*argString=None*)
The main function of the module.

Parameters **argString** (*list of strings*) – the options.

These are the steps:

1. Prints the options.
2. Runs Plink with the `mind` option (`runPlink()`).

`Step4.sample_missingness.`**`parseArgs`**(*argString=None*)
Parses the command line options and arguments.

Parameters **argString** (*list of strings*) – the options.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--ifile`	string	The input file prefix (either a Plink binary file or a tfile).
`--is-bfile`	bool	The input file (`--ifile`) is a bfile instead of a tfile.
`--mind`	float	The missingness threshold.
`--out`	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see `checkArgs()`).

`Step4.sample_missingness.`**`runPlink`**(*options*)
Run Plink with the `mind` option.

Parameters **options** (*argparse.Namespace*) – the options.

5.6 Marker Missingness Module

The usage of the standalone module is shown below:

```
$ pyGenClean_snp_missingness --help
usage: pyGenClean_snp_missingness [-h] --bfile FILE [--geno FLOAT]
                                  [--out FILE]

Computes sample missingness using Plink

optional arguments:
  -h, --help    show this help message and exit

Input File:
  --bfile FILE  The input file prefix (will find the plink binary files by
                appending the prefix to the .bim, .bed and .fam files,
                respectively.

Options:
  --geno FLOAT  The missingness threshold (remove SNPs with more than x
                percent missing genotypes). [Default: 0.020]

Output File:
  --out FILE    The prefix of the output files. [default: clean_geno]
```

5.6.1 Input FIles

This module uses PLINK's binary file format (`bed`, `bim` and `fam` files) for the source data set (the data of interest).

5.6.2 Procedure

Here are the steps performed by the module:

1. Runs Plink to remove markers with a missing rate above a user defined threshold.
2. Finds the markers that were removed (those that have a missing rate above the user defined threshold.

5.6.3 Output Files

The output files of each of the steps described above are as follow (note that the output prefix shown is the one by default [*i.e.* `clean_geno`]):

1. One set of Plink output files:
   - `clean_geno.fam`: the dataset with markers having a high missing rate removed (according to a user defined threshold).
2. One custom file:
   - `clean_geno.removed_snps`: a list of markers that have a high missing rate (above a user defined threshold).

5.6.4 The Algorithm

For more information about the actual algorithms and source codes (the `Step5.snp_missingness` module), refer to the following sections.

Step5.snp_missingness

exception `Step5.snp_missingness.`**`ProgramError`**(*msg*)
An `Exception` raised in case of a problem.

Parameters **msg** (*string*) – the message to print to the user before exiting.

`Step5.snp_missingness.`**`checkArgs`**(*args*)
Checks the arguments and options.

Parameters **args** (*argparse.Namespace*) – an object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the `ProgramError` class, a message is printed to the `sys.stderr` and the program exists with code 1.

`Step5.snp_missingness.`**`compareBIM`**(*args*)
Compare two BIM file.

Parameters **args** (*argparse.Namespace*) – the options.

Creates a *Dummy* object to mimic an `argparse.Namespace` class containing the options for the `PlinkUtils.compare_bim` module.

`Step5.snp_missingness.`**`main`**(*argString=None*)
The main function of the module.

Parameters **argString** (*list of strings*) – the options.

These are the steps:

1. Prints the options.
2. Runs Plink with the `geno` option (`runPlink()`).
3. Compares the two `bim` files (before and after the Plink `geno` analysis) (`compareBIM()`).

`Step5.snp_missingness.`**`parseArgs`**(*argString=None*)
Parses the command line options and arguments.

Parameters **argString** (*list of strings*) – the options.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--bfile`	string	The input file prefix (Plink binary file).
`--geno`	float	The missingness threshold.
`--out`	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see `checkArgs()`).

`Step5.snp_missingness.`**`runPlink`**(*options*)
Runs Plink with the geno option.

Parameters **options** (*argparse.Namespace*) – the options.

5.7 Sex Check Module

The usage of the standalone module is shown below:

```
$ pyGenClean_sex_check --help
usage: pyGenClean_sex_check [-h] --bfile FILE [--femaleF FLOAT]
                            [--maleF FLOAT] [--nbChr23 INT] [--gender-plot]
                            [--sex-chr-intensities FILE]
                            [--gender-plot-format FORMAT] [--lrr-baf]
                            [--lrr-baf-raw-dir DIR] [--lrr-baf-format FORMAT]
                            [--out FILE]

Check sex using Plink

optional arguments:
  -h, --help            show this help message and exit

Input File:
  --bfile FILE          The input file prefix (will find the Plink binary
                        files by appending the prefix to the .bed, .bim, and
                        .fam files, respectively.

Options:
  --femaleF FLOAT       The female F threshold. [default: < 0.300000]
  --maleF FLOAT         The male F threshold. [default: > 0.700000]
  --nbChr23 INT         The minimum number of markers on chromosome 23 before
                        computing Plink's sex check [default: 50]

Gender Plot:
  --gender-plot         Create the gender plot (summarized chr Y intensities
                        in function of summarized chr X intensities) for
                        problematic samples.
  --sex-chr-intensities FILE
                        A file containing alleles intensities for each of the
                        markers located on the X and Y chromosome for the
                        gender plot.
  --gender-plot-format FORMAT
                        The output file format for the gender plot (png, ps,
                        pdf, or X11 formats are available). [default: png]

LRR and BAF Plot:
  --lrr-baf             Create the LRR and BAF plot for problematic samples
  --lrr-baf-raw-dir DIR
                        Directory or list of directories containing
                        information about every samples (BAF and LRR).
  --lrr-baf-format FORMAT
                        The output file format for the LRR and BAF plot (png,
                        ps, pdf, or X11 formats are available). [default: png]

Output File:
  --out FILE            The prefix of the output files (which will be a Plink
                        binary file. [default: sexcheck]
```


cl

5.7.1 Input Files

This module uses PLINK's binary file format (`bed`, `bim` and `fam` files) for the source data set (the data of interest).

If the option of generating the gender plot is used, a file containing intensities information about each markers on the sexual chromosomes is required. This file (which could be gzipped) should contain at least the following columns:

- `Sample ID`: the unique sample id for each individual.
- `SNP Name`: the unique name of each markers.
- `Chr`: the name of the chromosome on which each marker is located.
- `X`: the intensities of the first allele of each marker.
- `Y`: the intensities of the second allele of each marker.

If the options of generating the BAF and LRR values is used, the name of a directory containing intensities file for each sample is required. The name of each file should be the unique sample id. This file (which could be gzipped) should contain at least the following columns:

- `Chr`: the name of the chromosome on which each marker is located.
- `Position`: the position of the marker on the chromosome.
- `B Allele Freq`: the BAF value of each marker.
- `Log R Ratio`: the LRR value of each marker.

If the two plotting module is used alone, one more file is required per module: a list of samples with gender problem and explanation for `Step6.gender_plot`, and only the list of samples with gender problem for `Step6.baf_lrr_plot` (both files are provided by the `Step6.sex_check` module).

5.7.2 Procedure

Here are the steps performed by the module:

1. Checks if there are enough markers on the chromosome 23. If not, the module stops here.
2. Runs the sex check analysis using Plink.
3. If there are no sex problems, the module quits.
4. Creates the recoded file for the chromosome 23.
5. Computes the heterozygosity percentage on the chromosome 23.
6. If there are enough markers on chromosome 24 (at least 1), creates the recoded file for this chromosome.
7. Computes the number of no call on the chromosome 24.
8. If required, plots the gender plot.
   (a) If there are `summarized_intensities` provided, reads the files and skips to step vi.
   (b) Reads the `bim` file to get markers on the sexual chromosomes.
   (c) Reads the `fam` file to get individual's gender.
   (d) Reads the file containing samples with sex problems.
   (e) Reads the intensities and summarizes them.
   (f) Plots the summarized intensities.
9. If required, plots the BAF and LRR plots.

(a) Reads the problematic samples.

(b) Finds and checks the raw files for each of the problematic samples.

(c) Plots the BAF and LRR plots.

5.7.3 Output Files

The output files of each of the steps described above are as follow (note that the output prefix shown is the one by default [*i.e* `sexcheck`]):

1. No output files created.
2. One set of PLINK's result files:
   - `sexcheck`: the result of the sex check procedure from Plink.
3. Two files are created if there are sex problems:
   - `sexcheck.list_problem_sex`: a summary of samples with sex problem.
   - `sexcheck.list_problem_sex_ids`: the list of sample ids with sex problem.
4. One set of Plink's files:
   - `sexcheck.chr23_recodeA`: the recoded file for the chromosome 23.
5. One custom output file:
   - `sexcheck.chr23_recodeA.raw.hetero`: the heterozygosity percentage on the chromosome 23. The file includes the following columns: `PED` (the pedigree ID), `ID` (the individual ID), `SEX` (the gender) and `HETERO` (the heterozygosity).
6. One set of Plink's files:
   - `sexcheck.chr24_recodeA`: the recoded file for the chromosome 24.
7. One custom output file:
   - `sexcheck.chr24_recodeA.raw.noCall`: the number of no call on the chromosome 24. The file includes the following columns: `PED` (the pedigree ID), `ID` (the individual ID), `SEX` (the gender), `nbGeno` (the number of genotypes on the chromosome 24) and `nbNoCall` (the number of genotypes that were not called on chromosome 24).
8. Multiple files and one plot. The files are created so that the plot can be generated again with different parameters (since the summarized intensities for each sample are really long to compute).
   - `sexcheck.png`: the gender plot (see Figure *Gender plot*).
   - `sexcheck.ok_females.txt`: the list of females without sex problem, including their summarized intensities on chromosome 23 and 24.
   - `sexcheck.ok_males.txt`: the list of males without sex problem, including their summarized intensities on chromosome 23 and 24.
   - `sexcheck.ok_unknowns.txt`: the list of unknown gender without sex problem, including their summarized intensities on chromosome 23 and 24.
   - `sexcheck.problematic_females.txt`: the list of females with sex problem, including their summarized intensities on chromosome 23 and 24.
   - `sexcheck.problematic_males.txt`: the list of males with sex problem, including their summarized intensities on chromosome 23 and 24.

- `sexcheck.problematic_unknowns.txt`: the list of unknown gender with sex problem, including their summarized intensities on chromosome 23 and 24. When this file is created by the `Step6.sex_check` module, it is empty.

9. A directory containing one file per individual with gender problem (see Figure *BAF and LRR plot*).

5.7.4 The Plots

The plot generated by the `Step6.gender_plot` module (the *Gender plot* figure) represents the summarized intensities of each sample of the data set. The color code represent the gender of each sample. Blue and red dots represent the males and females without gender problem, respectively. The green and purple triangles represent the males and females with gender problem. The gray dots and triangles represent the samples with unknown gender, with and without gender problem, respectively. This plot makes possible to find samples with sexual chromosomes abnormalities, such as males which are XXY or females with only one copy of the X chromosome (X0). Males that appear to be females and vice versa might in fact be sample mix up and would require further analysis.

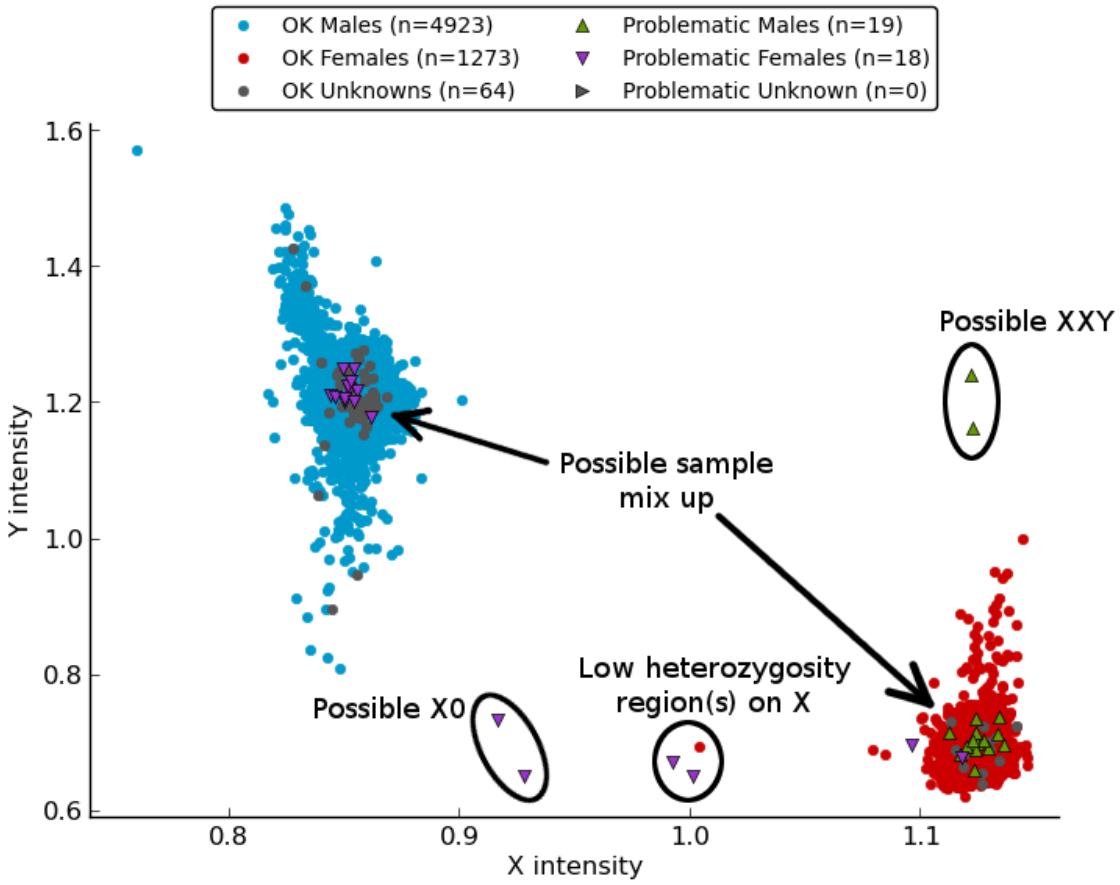


Figure 5.3: Gender plot

The *Gender plot* figure can also be manually created after the data clean up pipeline, using its results and this following standalone script:

```
$ pyGenClean_gender_plot --help
usage: pyGenClean_gender_plot [-h] --bfile FILE [--intensities FILE]
```

```
                              [--summarized-intensities FILE] --sex-problems
                              FILE [--format FORMAT] [--xlabel STRING]
                              [--ylabel STRING] [--out FILE]

Plots the gender using X and Y chromosomes' intensities

optional arguments:
  -h, --help            show this help message and exit

Input File:
  --bfile FILE          The plink binary file containing information about
                        markers and individuals. Must be specified if
                        '--summarized-intensities' is not.
  --intensities FILE    A file containing alleles intensities for each of the
                        markers located on the X and Y chromosome. Must be
                        specified if '--summarized-intensities' is not.
  --summarized-intensities FILE
                        The prefix of six files (prefix.ok_females.txt,
                        prefix.ok_males.txt, prefix.ok_unknowns.txt,
                        problematic_females.txt, problematic_males.txt and
                        problematic_unknowns.txt) containing summarized chr23
                        and chr24 intensities. Must be specified if '--bfile'
                        and '--intensities' are not.
  --sex-problems FILE   The file containing individuals with sex problems.
                        This file is not read if the option 'summarized-
                        intensities' is used.

Options:
  --format FORMAT       The output file format (png, ps, pdf, or X11 formats
                        are available). [default: png]
  --xlabel STRING       The label of the X axis. [default: X intensity]
  --ylabel STRING       The label of the Y axis. [default: Y intensity]

Output File:
  --out FILE            The prefix of the output files (which will be a Plink
                        binary file. [default: sexcheck]
```

The log R ratio (LRR) for a sample is the log ratio of the normalized R value for the marker divided by the expected normalized R value. Hence, a value of 0 means 2 copies. A drop in the LRR shows a loss of a copy, while an increasing LRR shows a gain of a copy. Expected LRR values on chromosome 23 for female and female are 0 and [-0.5, -1], respectively. The LRR values of each markers on both the X and Y chromosomes are shown in the *BAF and LRR plot* figure.

The B allele frequency (BAF) for a sample shows the theta value for a marker, corrected for cluster position. For a normal number of copies, markers should have a BAF around 1 (homozygous for the B allele), 0.5 (heterozygous) or 0 (homozygous for A allele). Normal females should have the three lines across the chromosome. Normal males should only have two lines, located near 1 or 0. The BAF values of each markers on both the X and Y chromosomes are shown in the *BAF and LRR plot* figure.

The *BAF and LRR plot* figure can also be manually created after the data clean up pipeline, using this following standalone script:

```
$ pyGenClean_baf_lrr_plot --help
usage: pyGenClean_baf_lrr_plot [-h] --problematic-samples FILE
                               [--use-full-ids] [--full-ids-delimiter CHAR]
                               --raw-dir DIR [--format FORMAT] [--out FILE]

Plots the BAF and LRR of problematic samples.
```

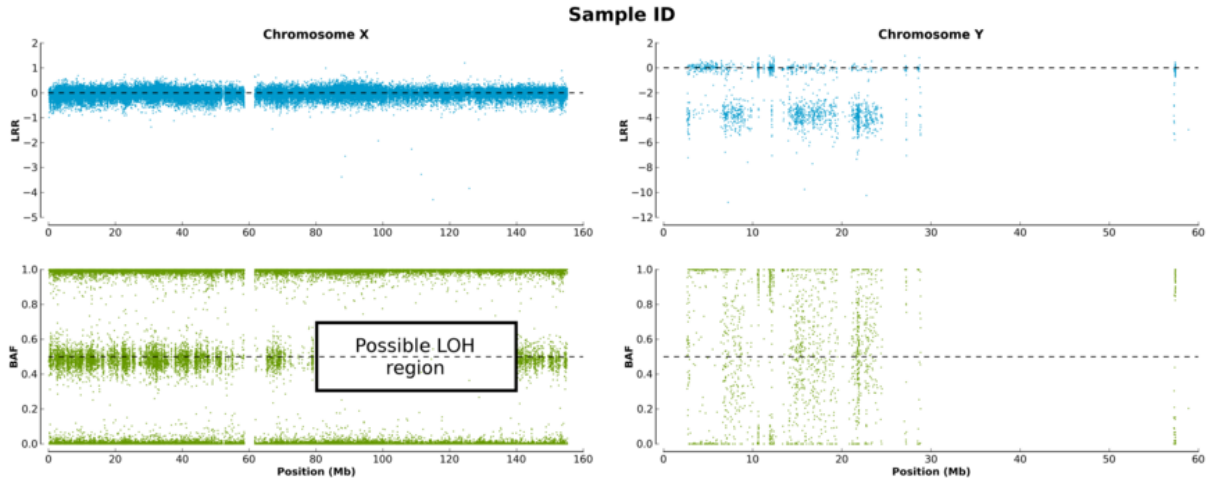


Figure 5.4: BAF and LRR plot

```
optional arguments:
  -h, --help            show this help message and exit

Input File:
  --problematic-samples FILE
                        A file containing the list of samples with sex
                        problems (family and individual ID required, separated
                        by a single tabulation). Uses only the individual ID
                        by default, unless --use-full-ids is used.
  --use-full-ids        Use this options to use full sample IDs (famID and
                        indID). Otherwise, only the indID will be use.
  --full-ids-delimiter CHAR
                        The delimiter between famID and indID for the raw file
                        names. [default: _]
  --raw-dir DIR         Directory containing information about every samples
                        (BAF and LRR).

Options:
  --format FORMAT       The output file format (png, ps, pdf, or X11 formats
                        are available). [default: png]

Output File:
  --out FILE            The prefix of the output files. [default: sexcheck]
```

5.7.5 The Algorithm

For more information about the actual algorithms and source codes (the `Step6.sex_check`, `Step6.gender_plot` and `Step6.baf_lrr_plot` modules), refer to the following sections.

Step6.sex_check

exception `Step6.sex_check.`**`ProgramError`**(*msg*)
An `Exception` raised in case of a problem.

Parameters **msg** (*string*) – the message to print to the user before exiting.

Step6.sex_check.**checkArgs**(*args*)
Checks the arguments and options.

Parameters **args** (*argparse.Namespace*) – an object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the `ProgramError` class, a message is printed to the `sys.stderr` and the program exists with code 1.

Step6.sex_check.**checkBim**(*fileName*, *minNumber*, *chromosome*)
Checks the BIM file for chrN markers.

Parameters

- **fileName** (*string*) –
- **minNumber** (*int*) –
- **chromosome** (*string*) –

Returns `True` if there are at least `minNumber` markers on chromosome `chromosome`, `False` otherwise.

Step6.sex_check.**computeHeteroPercentage**(*fileName*)
Computes the heterozygosity percentage.

Parameters **fileName** (*string*) – the name of the input file.

Reads the `ped` file created by Plink using the `recodeA` options (see `createPedChr23UsingPlink()`) and computes the heterozygosity percentage on the chromosome `23`.

Step6.sex_check.**computeNoCall**(*fileName*)
Computes the number of no call.

Parameters **fileName** (*string*) – the name of the file

Reads the `ped` file created by Plink using the `recodeA` options (see `createPedChr24UsingPlink()`) and computes the number and percentage of no calls on the chromosome `24`.

Step6.sex_check.**createGenderPlot**(*bfile*, *intensities*, *problematic_samples*, *format*, *out_prefix*)
Creates the gender plot.

Parameters

- **bfile** (*string*) – the prefix of the input binary file.
- **intensities** (*string*) – the file containing the intensities.
- **problematic_samples** (*string*) – the file containing the problematic samples.
- **format** (*string*) – the format of the output plot.
- **out_prefix** (*string*) – the prefix of the output file.

Creates the gender plot of the samples using the `Step6.gender_plot` module.

Step6.sex_check.**createLrrBafPlot**(*raw_dir*, *problematic_samples*, *format*, *out_prefix*)
Creates the LRR and BAF plot.

Parameters

- **raw_dir** (*string*) – the directory containing the intensities.
- **problematic_samples** (*string*) – the file containing the problematic samples.

- **format** (*string*) – the format of the plot.
- **out_prefix** (*string*) – the prefix of the output file.

Creates the LRR (Log R Ratio) and BAF (B Allele Frequency) of the problematic samples using the `Step6.baf_lrr_plot` module.

`Step6.sex_check.`**`createPedChr23UsingPlink`**(*options*)
Run Plink to create a ped format.

Parameters **options** (*argparse.Namespace*) – the options.

Uses Plink to create a `ped` file of markers on the chromosome `23`. It uses the `recodeA` options to use additive coding. It also subsets the data to keep only samples with sex problems.

`Step6.sex_check.`**`createPedChr24UsingPlink`**(*options*)
Run plink to create a ped format.

Parameters **options** (*argparse.Namespace*) – the options.

Uses Plink to create a `ped` file of markers on the chromosome `24`. It uses the `recodeA` options to use additive coding. It also subsets the data to keep only samples with sex problems.

`Step6.sex_check.`**`main`**(*argString=None*)
The main function of the module.

Parameters **argString** (*list of strings*) – the options.

These are the following steps:

1. Prints the options.
2. Checks if there are enough markers on the chromosome `23` (`checkBim()`). If not, quits here.
3. Runs the sex check analysis using Plink (`runPlinkSexCheck()`).
4. If there are no sex problems, then quits (`readCheckSexFile()`).
5. Creates the recoded file for the chromosome `23` (`createPedChr23UsingPlink()`).
6. Computes the heterozygosity percentage on the chromosome `23` (`computeHeteroPercentage()`).
7. If there are enough markers on chromosome `24` (at least 1), creates the recoded file for this chromosome (`createPedChr24UsingPlink()`).
8. Computes the number of no call on the chromosome `24` (`computeNoCall()`).
9. If required, plots the gender plot (`createGenderPlot()`).
10. If required, plots the BAF and LRR plot (`createLrrBafPlot()`).

`Step6.sex_check.`**`parseArgs`**(*argString=None*)
Parses the command line options and arguments.

Parameters **argString** (*list of strings*) – the options.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--bfile`	string	The input file prefix (Plink binary).
`--femaleF`	float	The female F threshold.
`--maleF`	float	The male F threshold.
`--nbChr23`	int	The minimum number of markers on chromosome 23 before computing Plink's sex check.
`--gender-plot`	bool	Create the gender plot.
`--sex-chr-intensities`	string	A file containing alleles intensities for each of the markers located on the X and Y chromosome.
`--gender-plot-format`	string	The output file format for the gender plot.
`--lrr-baf`	bool	Create the LRR and BAF plot.
`--lrr-baf-raw-dir`	string	Directory containing information about every samples (BAF and LRR).
`--lrr-baf-format`	string	The output file format.
`--out`	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see `checkArgs()`).

`Step6.sex_check.`**`readCheckSexFile`**(*fileName*, *allProblemsFileName*, *idsFileName*, *femaleF*, *maleF*)

Reads the Plink check-sex output file.

Parameters

- **fileName** (*string*) – the name of the input file.
- **allProblemsFileName** (*string*) – the name of the output file that will contain all the problems.
- **idsFileName** (*string*) – the name of the output file what will contain samples with sex problems.
- **femaleF** (*float*) – the F threshold for females.
- **maleF** (*float*) – the F threshold for males.

Returns `True` if there are sex problems, `False` otherwise.

Reads sex check file provided by `runPlinkSexCheck()` (Plink) and extract the samples that have sex problems.

`Step6.sex_check.`**`runCommand`**(*command*)

Run a command.

Parameters **command** (*list of strings*) – the command to run.

Tries to run a command. If it fails, raise a `ProgramError`. This function uses the `subprocess` module.

Warning: The variable command should be a list of strings (no other type).

`Step6.sex_check.`**`runPlinkSexCheck`**(*options*)

Runs Plink to perform a sex check analysis.

Parameters **options** (*argparse.Namespace*) – the options.

Uses Plink to perform a sex check analysis.

Step6.gender_plot

exception Step6.gender_plot.**ProgramError**(*msg*)
An Exception raised in case of a problem.

Parameters **msg** (*string*) – the message to print to the user before exiting.

Step6.gender_plot.**checkArgs**(*args*)
Checks the arguments and options.

Parameters **args** (*argparse.Namespace*) – an object containing the options of the program.

Returns True if everything was OK.

If there is a problem with an option, an exception is raised using the ProgramError class, a message is printed to the sys.stderr and the program exists with code 1.

Step6.gender_plot.**encode_chr**(*chromosome*)
Encodes chromosomes.

Parameters **chromosome** (*string*) – the chromosome to encode.

Returns the encoded chromosome as int.

It changes X, Y, XY and MT to 23, 24, 25 and 26, respectively. It changes everything else as int.

If ValueError is raised, then ProgramError is also raised. If a chromosome as a value below 1 or above 26, a ProgramError is raised.

```
>>> [encode_chr(str(i)) for i in range(0, 11)]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> [encode_chr(str(i)) for i in range(11, 21)]
[11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
>>> [encode_chr(str(i)) for i in range(21, 27)]
[21, 22, 23, 24, 25, 26]
>>> [encode_chr(i) for i in ["X", "Y", "XY", "MT"]]
[23, 24, 25, 26]
>>> encode_chr("27")
Traceback (most recent call last):
    ...
ProgramError: 27: invalid chromosome
>>> encode_chr("XX")
Traceback (most recent call last):
    ...
ProgramError: XX: invalid chromosome
```

Step6.gender_plot.**encode_gender**(*gender*)
Encodes the gender.

Parameters **gender** (*string*) – the gender to encode.

Returns the encoded gender.

It changes 1 and 2 to Male and Female respectively. It encodes everything else to Unknown.

```
>>> encode_gender("1")
'Male'
>>> encode_gender("2")
'Female'
>>> encode_gender("0")
'Unknown'
>>> encode_gender("This is not a gender code")
'Unknown'
```

Step6.gender_plot.**main**(*argString=None*)
The main function of the module.

Parameters **argString** (*list of strings*) – the options.

These are the steps:

1. Prints the options.
2. If there are `summarized_intensities` provided, reads the files (`read_summarized_intensities()`) and skips to step 7.
3. Reads the `bim` file to get markers on the sexual chromosomes (`read_bim()`).
4. Reads the `fam` file to get gender (`read_fam()`).
5. Reads the file containing samples with sex problems (`read_sex_problems()`).
6. Reads the intensities and summarizes them (`read_intensities()`).
7. Plots the summarized intensities (`plot_gender()`).

Step6.gender_plot.**parseArgs**(*argString=None*)
Parses the command line options and arguments.

Parameters **argString** (*list of strings*) – the options.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--bfile`	string	The plink binary file containing information about markers and individuals.
`--intensities`	string	A file containing alleles intensities for each of the markers located on the X and Y chromosome.
`--summarized-intensities`	string	The prefix of six files containing summarized chr23 and chr24 intensities.
`--sex-problems`	string	The file containing individuals with sex problems.
`--format`	string	The output file format (png, ps, pdf, or X11).
`--xlabel`	string	The label of the X axis.
`--ylabel`	string	The label of the Y axis.
`--out`	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see `checkArgs()`).

Step6.gender_plot.**plot_gender**(*data*, *options*)
Plots the gender.

Parameters

- **data** (*numpy.recarray*) – the data to plot.
- **options** (*argparse.Namespace*) – the options.

Plots the summarized intensities of the markers on the Y chromosomes in function of the markers on the X chromosomes, with problematic samples with different colors.

Also uses `print_data_to_file()` to save the data, so that it is faster to rerun the analysis.

Step6.gender_plot.**print_data_to_file**(*data*, *file_name*)
Prints data to file.

Parameters

clx

- **data** (*numpy.recarray*) – the data to print.
- **file_name** (*string*) – the name of the output file.

Step6.gender_plot.**read_bim**(*file_name*)
Reads the BIM file to gather marker names.

Parameters **file_name** (*string*) – the name of the `bim` file.

Returns a `dict` containing the chromosomal location of each marker on the sexual chromosomes.

It uses the `encode_chr()` to encode the chromosomes from `X` and `Y` to `23` and `24`, respectively.

Step6.gender_plot.**read_fam**(*file_name*)
Reads the FAM file to gather sample names.

Parameters **file_name** (*string*) – the `fam` file to read.

Returns a `dict` containing the gender of each samples.

It uses the `encode_gender()` to encode the gender from `1``and ``2` to `Male` and `Female`, respectively.

Step6.gender_plot.**read_intensities**(*file_name*, *needed_markers_chr*, *needed_samples_gender*, *sex_problems*)
Reads the intensities from a file.

Parameters

- **file_name** (*string*) – the name of the input file.
- **needed_markers_chr** (*dict*) – the markers that are needed.
- **needed_samples_gender** (*dict*) – the gender of all the samples.
- **sex_problems** (*frozenset*) – the sample with sex problem.

Returns a :py:class‘numpy.recarray‘ containing the following columns (for each of the samples): `sampleID`, `chr23`, `chr24`, `gender` and `status`.

Reads the normalized intensities from a final report. The file must contain the following columns: `SNP Name`, `Sample ID`, `X`, `Y` and `Chr`. It then keeps only the required markers (those that are on sexual chromosomes (`23` or `24`), encoding *NaN* intensities to zero.

The final data set contains the following information for each sample:

- `sampleID`: the sample ID.
- `chr23`: the summarized intensities for chromosome 23.
- `chr24`: the summarized intensities for chromosome 24.
- `gender`: the gender of the sample (`Male` or `Female`).
- `status`: the status of the sample (`OK` or `Problem`).

The summarized intensities for a chromosome (S_{chr}) is computed using this formula (where I_{chr} is the set of all marker intensities on chromosome chr):

$$S_{chr} = \frac{\sum I_{chr}}{||I_{chr}||}$$

Step6.gender_plot.**read_sex_problems**(*file_name*)
Reads the sex problem file.

Parameters **file_name** (*string*) – the name of the file containing sex problems.

Returns a `frozenset` containing samples with sex problem.

If there is no `file_name` (*i.e.* is `None`), then an empty `frozenset` is returned.

`Step6.gender_plot.`**`read_summarized_intensities`**(*prefix*)
Reads the summarized intensities from 6 files.

Parameters **prefix** (*string*) – the prefix of the six files.

Returns a :py:class‘numpy.recarray‘ containing the following columns (for each of the samples): `sampleID`, `chr23`, `chr24`, `gender` and `status`.

Instead of reading a final report (like `read_intensities()`), this function reads six files previously created by this module to gather sample information. Here are the content of the six files:

- `prefix.ok_females.txt`: information about females without sex problem.
- `prefix.ok_males.txt`: information about males without sex problem.
- **`prefix.ok_unknowns.txt`: information about unknown gender without sex** problem.
- **`prefix.problematic_females.txt`: information about females with sex** problem.
- **`prefix.problematic_males.txt`: information about males with sex** problem.
- **`prefix.problematic_unknowns.txt`: information about unknown gender with** sex problem.

Each file contains the following columns: `sampleID`, `chr23`, `chr24`, `gender` and `status`.

The final data set contains the following information for each sample:

- `sampleID`: the sample ID.
- `chr23`: the summarized intensities for chromosome 23.
- `chr24`: the summarized intensities for chromosome 24.
- `gender`: the gender of the sample (`Male` or `Female`).
- `status`: the status of the sample (`OK` or `Problem`).

The summarized intensities for a chromosome (S_{chr}) is computed using this formula (where I_{chr} is the set of all marker intensities on chromosome chr):

$$S_{chr} = \frac{\sum I_{chr}}{||I_{chr}||}$$

Step6.baf_lrr_plot

exception `Step6.baf_lrr_plot.`**`ProgramError`**(*msg*)
An `Exception` raised in case of a problem.

Parameters **msg** (*string*) – the message to print to the user before exiting.

`Step6.baf_lrr_plot.`**`checkArgs`**(*args*)
Checks the arguments and options.

Parameters **args** (*argparse.Namespace*) – an object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the `ProgramError` class, a message is printed to the `sys.stderr` and the program exists with code 1.

Step6.baf_lrr_plot.**check_file_names**(*samples*, *raw_dir*, *options*)
Check if all files are present.

Parameters

- **samples** (*list of tuples*) – a list of tuples with the family ID as first element (string) and sample ID as last element (string).
- **raw_dir** (*string*) – the directory containing the raw files.
- **options** (*argparse.Namespace*) – the options.

Returns a dict containing samples as key (a tuple with the family ID as first element and sample ID as last element) and the name of the raw file as element.

Step6.baf_lrr_plot.**encode_chromosome**(*chromosome*)
Encodes chromosomes.

Parameters **chromosome** (*string*) – the chromosome to encode.

Returns the encoded chromosome.

Encodes the sexual chromosomes, from `23` and `24` to `X` and `Y`, respectively.

Note: Only the sexual chromosomes are encoded.

```
>>> encode_chromosome("23")
'X'
>>> encode_chromosome("24")
'Y'
>>> encode_chromosome("This is not a chromosome")
'This is not a chromosome'
```

Step6.baf_lrr_plot.**main**(*argString=None*)
The main function of this module.

Parameters **argString** (*list of strings*) – the options.

These are the steps:

1. Prints the options.
2. Reads the problematic samples (`read_problematic_samples()`).
3. Finds and checks the raw files for each of the problematic samples (`check_file_names()`).
4. Plots the BAF and LRR (`plot_baf_lrr()`).

Step6.baf_lrr_plot.**parseArgs**(*argString=None*)
Parses the command line options and arguments.

Parameters **argString** (*list of strings*) – the options.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--problematic-samples`	string	The list of sample with sex problems to plot
`--use-full-ids`	bool	Use full sample IDs (famID and indID).
`--full-ids-delimiter`	string	The delimiter between famID and indID.
`--raw-dir`	string	Directory containing information about every samples (BAF and LRR).
`--format`	string	The output file format (png, ps, pdf, or X11).
`--out`	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see `checkArgs()`).

`Step6.baf_lrr_plot.`**`plot_baf_lrr`**(*file_names*, *options*)
Plot BAF and LRR for a list of files.

Parameters

- **file_names** (*dict*) – contains the name of the input file for each sample.
- **options** (*argparse.Namespace*) – the options.

Plots the BAF (B Allele Frequency) and LRR (Log R Ratio) of each samples. Only the sexual chromosome are shown.

`Step6.baf_lrr_plot.`**`read_problematic_samples`**(*file_name*)
Reads a file with sample IDs.

Parameters **file_name** (*string*) – the name of the file containing problematic samples after sex check.

Returns a set of problematic samples (tuple containing the family ID as first element and the sample ID as last element).

Reads a file containing problematic samples after sex check. The file is provided by the module `Step6.sex_check`. This file contains two columns, the first one being the family ID and the second one, the sample ID.

5.8 Plate Bias Module

The usage of the standalone module is shown below:

```
$ pyGenClean_plate_bias --help
usage: pyGenClean_plate_bias [-h] --bfile FILE --loop-assoc FILE
                             [--pfilter FLOAT] [--out FILE]

Check for plate bias.

optional arguments:
  -h, --help          show this help message and exit

Input File:
  --bfile FILE        The input file prefix (will find the plink binary files
                      by appending the prefix to the .bim, .bed and .fam files,
                      respectively.
  --loop-assoc FILE   The file containing the plate organization of each
                      samples. Must contains three column (with no header):
                      famID, indID and plateName.

Options:
  --pfilter FLOAT     The significance threshold used for the plate effect.
                      [default: 1.0e-07]

Output File:
  --out FILE          The prefix of the output files. [default: plate_bias]
```

5.8.1 Input Files

This module uses PLINK's binary file format (`bed`, `bim` and `fam` files) for the source data set (the data of interest).

5.8.2 Procedure

Here are the steps performed by the module:

1. Runs the plate bias analysis using Plink.
2. Extracts the list of significant markers after plate bias analysis.
3. Computes the frequency of all significant markers after plate bias analysis.

5.8.3 Output Files

The output files of each of the steps described above are as follow (note that the output prefix shown is the one by default [*i.e.* `plate_bias`]):

1. One set of PLINK's result files:
   - `plate_bias`: this includes file like `plate_bias.PLATE_NAME.assoc.fisher` containing a list of markers that were significant after the plate bias analysis.
2. One file:
   - `plate_bias.significant_SNPs.txt`: a file containing a list of markers that were significant after the plate bias analysis (contained in all the `*.fisher` files).
3. One set of PLINK's result files:
   - `plate_bias.significant_SNPs`: the result files containing the frequency of all the markers that were significant after the plate bias analysis (contained in all the `*.fisher` files).

5.8.4 The Algorithm

For more information about the actual algorithms and source codes (the `Step7.plate_bias` module), refer to the following sections.

Step7.plate_bias

exception `Step7.plate_bias.`**`ProgramError`**(*msg*)
An `Exception` raised in case of a problem.

Parameters **msg** (*string*) – the message to print to the user before exiting.

`Step7.plate_bias.`**`checkArgs`**(*args*)
Checks the arguments and options.

Parameters **args** (*argparse.Namespace*) – an object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the `ProgramError` class, a message is printed to the `sys.stderr` and the program exists with code 1.

`Step7.plate_bias.`**`computeFrequencyOfSignificantSNPs`**(*options*)
Computes the frequency of the significant markers.

Parameters **options** (*argparse.Namespace*) – the options.

Extract a list of markers (significant after plate bias analysis) and computes their frequencies.

`Step7.plate_bias.`**`executePlateBiasAnalysis`**(*options*)
Execute the plate bias analysis with Plink.

Parameters **options** (*argparse.Namespace*) – the options.

`Step7.plate_bias.`**`extractSignificantSNPs`**(*prefix*)
Extract significant SNPs in the fisher file.

Parameters **prefix** (*string*) – the prefix of the input file.

Reads a list of significant markers (`prefix.assoc.fisher`) after plate bias analysis with Plink. Writes a file (`prefix.significant_SNPs.txt`) containing those significant markers.

`Step7.plate_bias.`**`main`**(*argString=None*)
The main function of this module.

Parameters **argString** (*list of strings*) – the options.

These are the steps:

1. Runs a plate bias analysis using Plink (`executePlateBiasAnalysis()`).
2. Extracts the list of significant markers after plate bias analysis (`extractSignificantSNPs()`).
3. Computes the frequency of all significant markers after plate bias analysis (`computeFrequencyOfSignificantSNPs()`).

`Step7.plate_bias.`**`parseArgs`**(*argString=None*)
Parses the command line options and arguments.

Parameters **argString** (*list of strings*) – the options.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--bfile`	string	The input file prefix (Plink binary).
`--loop-assoc`	string	The file containing the plate organization of each samples.
`--pfilter`	float	The significance threshold used for the plate effect.
`--out`	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see `checkArgs()`).

`Step7.plate_bias.`**`runCommand`**(*command*)
Run a command.

Parameters **command** (*list of strings*) – the command to run.

Tries to run a command. If it fails, raise a `ProgramError`. This function uses the `subprocess` module.

Warning: The variable `command` should be a list of strings (no other type).

5.9 Heterozygous Haploid Module

The usage of the standalone module is shown below:

```
$ pyGenClean_remove_heterozygous_haploid --help
usage: pyGenClean_remove_heterozygous_haploid [-h] --bfile FILE [--out FILE]

Removes heterozygous haploid genotypes.

optional arguments:
  -h, --help    show this help message and exit

Input File:
  --bfile FILE  The input file prefix (will find the plink binary files by
                appending the prefix to the .bim, .bed and .fam files,
                respectively.

Output File:
  --out FILE    The prefix of the output files. [default:
                without_hh_genotypes]
```

5.9.1 Input Files

This module uses PLINK's binary file format (`bed`, `bim` and `fam` files) for the source data set (the data of interest).

5.9.2 Procedure

Here are the steps performed by the module:

1. Uses Plink to remove the heterozygous haploid genotypes.

5.9.3 Output Files

The output files of each of the steps described above are as follow (note that the output prefix shown is the one by default [*i.e.* `without_hh_genotypes`]):

1. On set of Plink's output files:
    - `without_hh_genotypes`: the data set with heterozygous haploid genotypes removed.

5.9.4 The Algorithm

For more information about the actual algorithms and source codes (the `Step8.remove_heterozygous_haploid` module), refer to the following sections.

Step8.remove_heterozygous_haploid

exception `Step8.remove_heterozygous_haploid.`**`ProgramError`**(*msg*)
An `Exception` raised in case of a problem.

Parameters **msg** (*string*) – the message to print to the user before exiting.

`Step8.remove_heterozygous_haploid.`**`checkArgs`**(*args*)
Checks the arguments and options.

Parameters **args** (*argparse.Namespace*) – a an object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the `ProgramError` class, a message is printed to the `sys.stderr` and the program exists with code 1.

`Step8.remove_heterozygous_haploid.`**`main`**`(`*argString=None*`)`
The main function of this module.

Parameters **argString** (*list of strings*) – the options.

`Step8.remove_heterozygous_haploid.`**`parseArgs`**`(`*argString=None*`)`
Parses the command line options and arguments.

Parameters **argString** (*list of strings*) – the options.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--bfile`	string	The input file prefix (Plink binary file).
`--out`	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see `checkArgs()`).

`Step8.remove_heterozygous_haploid.`**`runPlink`**`(`*options*`)`
Sets heterozygous haploid markers to missing Plink.

Parameters **options** (*argparse.Namespace*) – the options.

5.10 Related Samples Module

The usage of the standalone module is shown below:

```
$ pyGenClean_find_related_samples --help
usage: pyGenClean_find_related_samples [-h] --bfile FILE [--genome-only]
                                       [--min-nb-snp INT]
                                       [--indep-pairwise STR STR STR]
                                       [--maf FLOAT] [--ibs2-ratio FLOAT]
                                       [--sge] [--line-per-file-for-sge INT]
                                       [--out FILE]

Finds related samples according to IBS values.

optional arguments:
  -h, --help            show this help message and exit

Input File:
  --bfile FILE          The input file prefix (will find the plink binary
                        files by appending the prefix to the .bim, .bed and
                        .fam files, respectively.)

Options:
  --genome-only         Only create the genome file
  --min-nb-snp INT      The minimum number of markers needed to compute IBS
                        values. [Default: 10000]
  --indep-pairwise STR STR STR
                        Three numbers: window size, window shift and the r2
                        threshold. [default: ['50', '5', '0.1']]
  --maf FLOAT           Restrict to SNPs with MAF >= threshold. [default:
```

```
                        0.05]
  --ibs2-ratio FLOAT    The initial IBS2* ratio (the minimum value to show in
                        the plot. [default: 0.8]
  --sge                 Use SGE for parallelization.
  --line-per-file-for-sge INT
                        The number of line per file for SGE task array.
                        [default: 100]

Output File:
  --out FILE            The prefix of the output files. [default: ibs]
```

5.10.1 Input Files

This module uses PLINK's binary file format (`bed`, `bim` and `fam` files) for the source data set (the data of interest).

5.10.2 Procedure

Here are the steps performed by the module:

1. Uses Plink to extract markers according to LD.
2. Checks if there is enough markers after pruning.
3. Extract markers according to LD.
4. Runs Plink with the `genome` option to compute the IBS values.
5. Finds related individuals and gets values for plotting.
6. Plots `Z1` in function of `IBS2 ratio` for related individuals.
7. Plots `Z2` in function of `IBS2 ratio` for related individuals.

5.10.3 Output Files

The output files of each of the steps described above are as follow (note that the output prefix shown is the one by default [*i.e.* `ibs`]):

1. One set of PLINK's result files:
   - `ibs.pruning_0.1`: the results of the pruning process of Plink. The value depends on the option of `--indep-pairwise`. The markers that are kept are in the file `ibs.pruning_0.1.prune.in`.
2. No file created.
3. One set of PLINK's binary files:
   - `ibs.pruned_data`: the data sets containing only the marker from the first step (the list is in `ibs.pruning_0.1.prune.in`).
4. One set of PLINK's result files (two if `--sge` is used):
   - `ibs.frequency`: PLINK's result files when computing the frequency of each of the pruned markers. This data set will exist only if the option `--sge` is used.
   - `ibs.genome`: PLINK's results including IBS values.
5. One file provided by the `Step9.find_related_samples` and three files provided by `Step9.merge_related_samples`:

- **ibs.related_individuals: a subset of the ibs.genome.genome file** containing only samples that are considered to be related. Three columns are appended to the original ibs.genome.genome file: IBS2_ratio (the value that is considered to find related individuals), status (the type of relatedness [*e.g.* twins]) and code (a numerical code that represent the status). This file is provided by the Step9.find_related_samples module.
- ibs.merged_related_individuals: a file aggregating related samples in groups, containing the following columns: index (the group number), FID1 (the family ID of the first sample), IID1 (the individual ID of the first sample), FID2 (the family ID of the second sample), IID2 (the individual ID of the second sample) and status (the type of relatedness between the two samples). This file is provided by the merge_related_samples.
- ibs.chosen_related_individuals: the related individuals that were randomly chosen from each group to be kept in the final data set. This file is provided by the merge_related_samples.
- ibs.discarded_related_individual: the related individuals that needs to be discarded, so that the final data set include only unrelated individuals. This file is provided by the merge_related_samples.

6. One image file:
   - ibs.related_individuals_z1.png: a plot showing the Z_1 value in function of the $IBS2^*_{ratio}$ for all samples above a certain $IBS2^*_{ratio}$ (the default threshold is 0.8). See Figure *Z1 in function of IBS2 ratio*.
7. One image file:
   - ibs.related_individuals_z2.png: a plot showing the Z_2 value in function of the $IBS2^*_{ratio}$ for all samples above a certain $IBS2^*_{ratio}$ (the default threshold is 0.8). See Figure *Z2 in function of IBS2 ratio*.

5.10.4 The Plots

The first plot (*Z1 in function of IBS2 ratio* figure) that is created is Z_1 in function of $IBS2^*_{ratio}$ (where each point represents a pair of related individuals. The color code comes from the different value of Z_0, Z_1 and Z_2, as described in the Step9.find_related_samples.extractRelatedIndividuals() function. In this plot, there are four locations where related samples tend to accumulate (first degree relatives (full sibs), second degree relatives (half-sibs, grand-parent-child or uncle-nephew), parent-child and twins (or duplicated samples). The unknown sample pairs represent possible undetected related individuals.

The second plot (*Z2 in function of IBS2 ratio* figure) that is created is Z_2 in function of $IBS2^*_{ratio}$ (where each point represents a pair of related individuals. It's just another representation of relatedness of sample pairs, where the location of the "clusters" is different.

5.10.5 The Algorithm

For more information about the actual algorithms and source codes (the Step9.find_related_samples and Step9.merge_related_samples modules), refer to the following sections.

Step9.find_related_samples

exception Step9.find_related_samples.**ProgramError**(*msg*)
An Exception raised in case of a problem.

Parameters **msg** (*string*) – the message to print to the user before exiting.

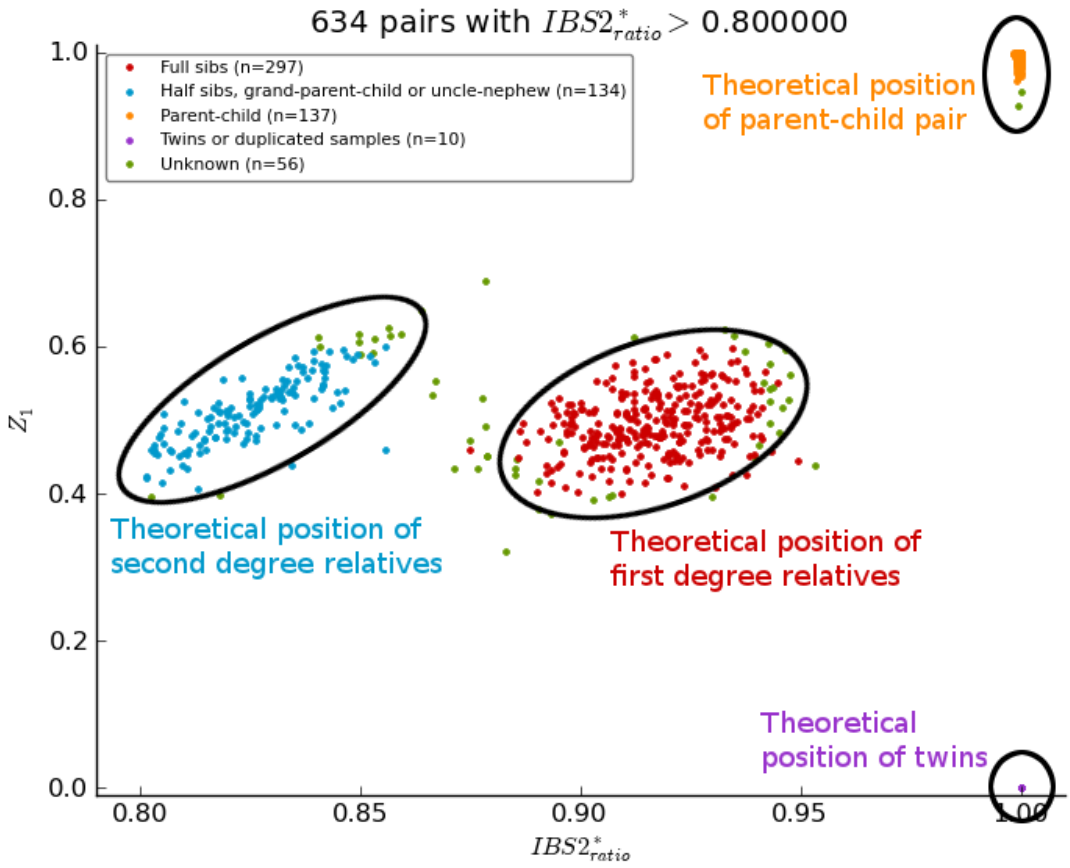


Figure 5.5: Z1 in function of IBS2 ratio

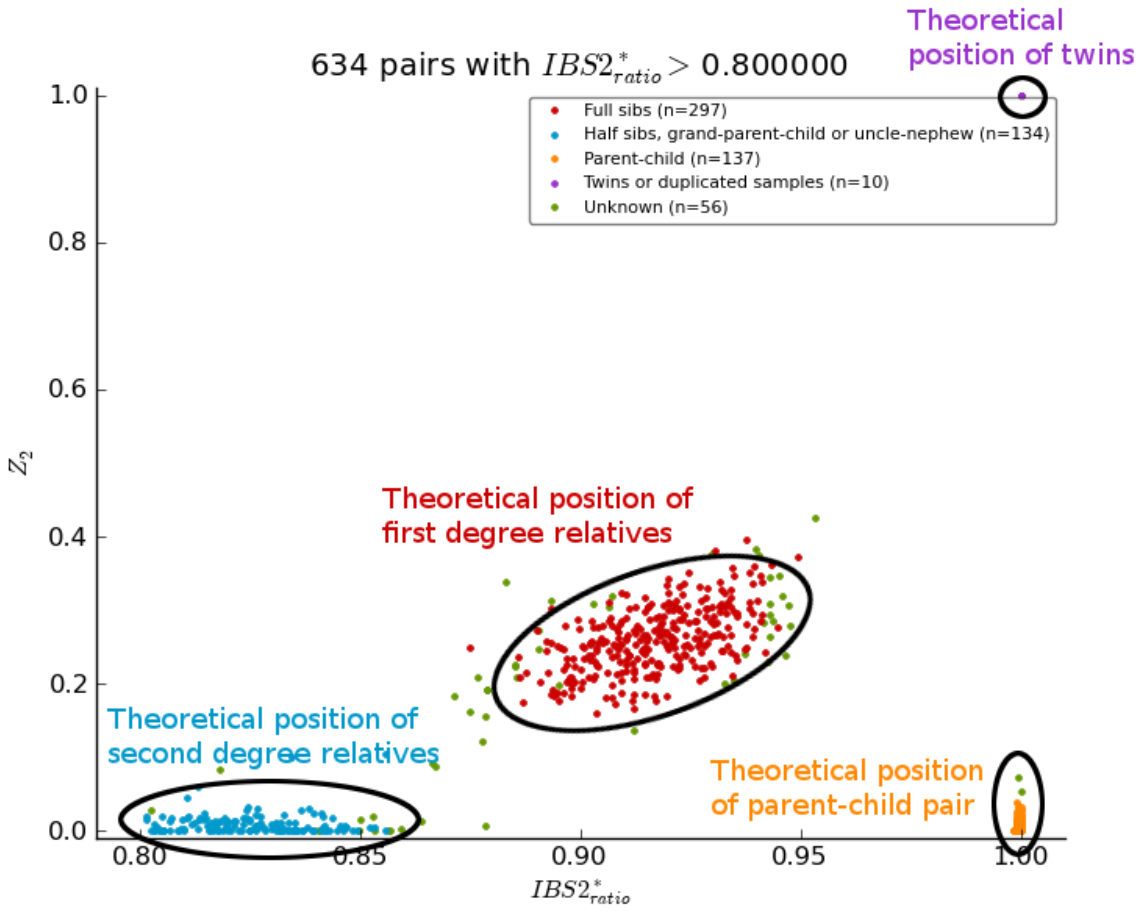


Figure 5.6: Z2 in function of IBS2 ratio

Step9.find_related_samples.**checkArgs**(*args*)
Checks the arguments and options.

Parameters **args** (*argparse.Namespace*) – an object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the `ProgramError` class, a message is printed to the `sys.stderr` and the program exists with code 1.

Step9.find_related_samples.**checkNumberOfSNP**(*fileName*, *minimumNumber*)
Check there is enough SNPs in the file (with minimum).

Parameters

- **fileName** (*string*) – the name of the file.
- **minimumNumber** (*int*) – the minimum number of markers that needs to be in the file.

Returns `True` if there is enough markers in the file, `False` otherwise.

Reads the number of markers (number of lines) in a file.

Step9.find_related_samples.**extractRelatedIndividuals**(*fileName*, *outPrefix*, *ibs2_ratio_threshold*)
Extract related individuals according IBS2* ratio.

Parameters

- **fileName** (*string*) – the name of the input file.
- **outPrefix** (*string*) – the prefix of the output files.
- **ibs2_ratio_threshold** (*float*) – the ibs2 ratio threshold (tells if sample pair is related or not).

Returns a `numpy.recarray` data set containing (for each related sample pair) the `ibs2 ratio`, `Z1`, `Z2` and the type of relatedness.

Reads a `genome` file (provided by `runGenome()`) and extract related sample pairs according to `IBS2 ratio`.

A `genome` file contains at least the following information for each sample pair:

•**FID1:** the family ID of the first sample in the pair.

•**IID1:** the individual ID of the first sample in the pair.

•**FID2:** the family ID of the second sample in the pair.

•**IID2:** the individual ID of the second sample in the pair.

•**Z0:** the probability that $IBD = 0$.

•**Z1:** the probability that $IBD = 1$.

•**Z2:** the probability that $IBD = 2$.

•**HOMHOM:** the number of $IBS = 0$ SNP pairs used in `PPC` test.

•**HETHET:** the number of $IBS = 2$ het/het SNP pairs in `PPC` test.

The `IBS2 ratio` is computed using the following formula:

$$\text{IBS2 ratio} = \frac{\text{HETHET}}{\text{HOMHOM} + \text{HETHET}}$$

If the `IBS2 ratio` is higher than the threshold, the samples in the pair are related. The following values help in finding the relatedness of the sample pair.

Values	Relation	Code
$0.17 \leq z_0 \leq 0.33$ and $0.40 \leq z_1 \leq 0.60$	Full-sibs	1
$0.40 \leq z_0 \leq 0.60$ and $0.40 \leq z_1 \leq 0.60$	Half-sibs or Grand-parent-Child or Uncle-Nephew	2
$z_0 \leq 0.05$ and $z_1 \geq 0.95$ and $z_2 \leq 0.05$	Parent-Child	3
$z_0 \leq 0.05$ and $z_1 \leq 0.05$ and $z_2 \geq 0.95$	Twins or Duplicated samples	4

`Step9.find_related_samples.`**`extractSNPs`**(*snpsToExtract*, *options*)
Extract markers using Plink.

Parameters

- **snpsToExtract** (*string*) – the name of the file containing markers to extract.
- **options** (*argparse.Namespace*) – the options

Returns the prefix of the output files.

`Step9.find_related_samples.`**`main`**(*argString=None*)
The main function of this module.

Parameters **argString** (*list of strings*) – the options.

Here are the steps for this function:

1. Prints the options.
2. Uses Plink to extract markers according to LD (`selectSNPsAccordingToLD()`).
3. Checks if there is enough markers after pruning (`checkNumberOfSNP()`). If not, then quits.
4. Extract markers according to LD (`extractSNPs()`).
5. Runs Plink with the `genome` option (`runGenome()`). Quits here if the user asker only for the `genome` file.
6. Finds related individuals and gets values for plotting (`extractRelatedIndividuals()`).
7. Plots `Z1` in function of `IBS2 ratio` for related individuals (`plot_related_data()`).
8. Plots `Z2` in function of `IBS2 ratio` for related individuals (`plot_related_data()`).

`Step9.find_related_samples.`**`mergeGenomeLogFiles`**(*outPrefix*, *nbSet*)
Merge genome and log files together.

Parameters

- **outPrefix** (*string*) – the prefix of the output files.
- **nbSet** (*int*) – The number of set of files to merge together.

Returns the name of the output file (the `genome` file).

After merging, the files are deleted to save space.

`Step9.find_related_samples.`**`parseArgs`**(*argString=None*)
Parses the command line options and arguments.

Parameters **argString** (*list of strings*) – the options.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options Type		Description
`--bfile`	string	The input file prefix (Plink binary file).
`--genome-only`	bool	Only create the genome file.
`--min-nb-snp`	int	The minimum number of markers needed to compute IBS values.
`--indep-pairwise`	string	Three numbers: window size, window shift and the r2 threshold.
`--maf`	string	Restrict to SNPs with MAF >= threshold.
`--ibs2-ratio`	float	The initial IBS2* ratio (the minimum value to show in the plot.
`--sge`	bool	Use SGE for parallelization.
`--line-per-file-for-sge`	int	The number of line per file for SGE task array.
`--out`	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see `checkArgs()`).

`Step9.find_related_samples.`**`plot_related_data`**(*x*, *y*, *code*, *ylabel*, *fileName*, *options*)

Plot Z1 and Z2 in function of IBS2* ratio.

Parameters

- **x** (*numpy.array of floats*) – the x axis of the plot (`IBS2 ratio`).
- **y** (*numpy.array of floats*) – the y axis of the plot (either `z1` or `z2`).
- **code** (*numpy.array of strings*) – the code of the relatedness of each sample pair.
- **ylabel** (*string*) – the label of the y axis (either `z1` or `z2`).
- **fileName** (*string*) – the name of the output file.
- **options** (*argparse.Namespace*) – the options.

There are four different relation codes (represented by 4 different color in the plots:

Code	Relation	Color
1	Full-sbis	`#CC0000`
2	Half-sibs or Grand-parent-Child or Uncle-Nephew	`#0099CC`
3	Parent-Child	`#FF8800`
4	Twins or Duplicated samples	`#9933CC`

Sample pairs with unknown relation are plotted using `#669900` as color.

`Step9.find_related_samples.`**`runCommand`**(*command*)

Run a command.

Parameters **command** (*list of strings*) – the command to run.

Tries to run a command. If it fails, raise a `ProgramError`. This function uses the `subprocess` module.

Warning: The variable `command` should be a list of strings (no other type).

`Step9.find_related_samples.`**`runGenome`**(*bfile*, *options*)

Runs the genome command from plink.

Parameters

- **bfile** (*string*) – the input file prefix.
- **options** (*argparse.Namespace*) – the options.

Returns the name of the `genome` file.

Runs Plink with the `genome` option. If the user asks for SGE (`options.sge` is True), a frequency file is first created by plink. Then, the input files are split in `options.line_per_file_for_sge` and Plink is called (using the `genome` option) on the cluster using SGE (`runGenomeSGE()`). After the analysis, Plink's output files and logs are merged using `mergeGenomeLogFiles()`.

`Step9.find_related_samples.`**`runGenomeSGE`**(*bfile*, *freqFile*, *nbJob*, *outPrefix*)
Runs the genome command from plink, on SGE.

Parameters

- **bfile** (*string*) – the prefix of the input file.
- **freqFile** (*string*) – the name of the frequency file (from Plink).
- **nbJob** (*int*) – the number of jobs to launch.
- **outPrefix** (*string*) – the prefix of all the output files.

Runs Plink with the `genome` options on the cluster (using SGE).

`Step9.find_related_samples.`**`selectSNPsAccordingToLD`**(*options*)
Compute LD using Plink.

Parameters **options** (*argparse.Namespace*) – the options.

Returns the name of the output file (from Plink).

`Step9.find_related_samples.`**`splitFile`**(*inputFileName*, *linePerFile*, *outPrefix*)
Split a file.

Parameters

- **inputFileName** (*string*) – the name of the input file.
- **linePerFile** (*int*) – the number of line per file (after splitting).
- **outPrefix** (*string*) – the prefix of the output files.

Returns the number of created temporary files.

Splits a file (`inputFileName` into multiple files containing at most `linePerFile` lines.

Step9.merge_related_samples

exception `Step9.merge_related_samples.`**`ProgramError`**(*msg*)
An `Exception` raised in case of a problem.

Parameters **msg** (*string*) – the message to print to the user before exiting.

`Step9.merge_related_samples.`**`checkArgs`**(*args*)
Checks the arguments and options.

Parameters **args** (*argparse.Namespace*) – a an object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the `ProgramError` class, a message is printed to the `sys.stderr` and the program exists with code 1.

`Step9.merge_related_samples.`**`main`**(*argString=None*)
The main function of the module.

Parameters **argString** (*list of strings*) – the options.

`Step9.merge_related_samples.`**`merge_related_samples`**(*file_name*, *out_prefix*, *no_status*)
Merge related samples.

Parameters

- **file_name** (*string*) – the name of the input file.
- **out_prefix** (*string*) – the prefix of the output files.
- **no_status** (*boolean*) – is there a status column in the file?

In the output file, there are a pair of samples per line. Hence, one can find related individuals by merging overlapping pairs.

`Step9.merge_related_samples.`**`parseArgs`**(*argString=None*)

Parses the command line options and arguments.

Parameters **argString** (*list of strings*) – the options.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--ibs-related`	string	The input file containing related individuals according to IBS value.
`--no-status`	bool	The input file doesn't have a `status` column.
`--out`	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see `checkArgs()`).

5.11 Ethnicity Module

The usage of the standalone module is shown below:

```
$ pyGenClean_check_ethnicity --help
usage: check_ethnicity.py [-h] --bfile FILE --ceu-bfile FILE --yri-bfile FILE
                          --jpt-chb-bfile FILE [--min-nb-snp INT]
                          [--indep-pairwise STR STR STR] [--maf FLOAT] [--sge]
                          [--line-per-file-for-sge INT] [--nb-components INT]
                          [--outliers-of POP] [--multiplier FLOAT]
                          [--xaxis COMPONENT] [--yaxis COMPONENT]
                          [--format FORMAT] [--title STRING] [--xlabel STRING]
                          [--ylabel STRING] [--out FILE]

Check samples' ethnicity using reference populations and IBS.

optional arguments:
  -h, --help            show this help message and exit

Input File:
  --bfile FILE          The input file prefix (will find the plink binary
                        files by appending the prefix to the .bim, .bed and
                        .fam files, respectively.
  --ceu-bfile FILE      The input file prefix (will find the plink binary
                        files by appending the prefix to the .bim, .bed and
                        .fam files, respectively.) for the CEU population
  --yri-bfile FILE      The input file prefix (will find the plink binary
                        files by appending the prefix to the .bim, .bed and
                        .fam files, respectively.) for the YRI population
  --jpt-chb-bfile FILE  The input file prefix (will find the plink binary
                        files by appending the prefix to the .bim, .bed and
```

```
                        .fam files, respectively.) for the JPT-CHB population

Options:
  --min-nb-snp INT      The minimum number of markers needed to compute IBS
                        values. [Default: 8000]
  --indep-pairwise STR STR STR
                        Three numbers: window size, window shift and the r2
                        threshold. [default: ['50', '5', '0.1']]
  --maf FLOAT           Restrict to SNPs with MAF >= threshold. [default:
                        0.05]
  --sge                 Use SGE for parallelization.
  --line-per-file-for-sge INT
                        The number of line per file for SGE task array.
                        [default: 100]
  --nb-components INT   The number of component to compute. [default: 10]

Outlier Options:
  --outliers-of POP     Finds the outliers of this population. [default: CEU]
  --multiplier FLOAT    To find the outliers, we look for more than x times
                        the cluster standard deviation. [default: 1.9]
  --xaxis COMPONENT     The component to use for the X axis. [default: C1]
  --yaxis COMPONENT     The component to use for the Y axis. [default: C2]

MDS Plot Options:
  --format FORMAT       The output file format (png, ps, pdf, or X11 formats
                        are available). [default: png]
  --title STRING        The title of the MDS plot. [default: C2 in function of
                        C1 - MDS]
  --xlabel STRING       The label of the X axis. [default: C1]
  --ylabel STRING       The label of the Y axis. [default: C2]

Output File:
  --out FILE            The prefix of the output files. [default: ethnicity]
```

5.11.1 Input Files

This module uses PLINK's binary file format (`bed`, `bim` and `fam` files) for the source data set (the data of interest) and three sets of binary files for the reference panels (CEU, YRI and JPG-CHB).

5.11.2 Procedure

Here are the steps performed by the module:

1. Finds the overlapping markers between the three reference panels and the source panel.
2. Extracts the required markers from all the data sets (source and reference panels).
3. Combines the three reference panels together to create a single data set.
4. Renames the reference panel's markers so that they match the names of the markers in the source panel.
5. Computes the frequency of all the markers from the reference and the source panels.
6. Finds the markers to flip in the reference panel, to enable fast comparison with the source panel.
7. Excludes markers that cannot be flip from the reference and the source panels.
8. Flips the markers that need to be in the reference panel.

9. Combines the reference and the source panels.
10. Computes the IBS values (see Section *Related Samples Module* for more information).
11. Creates the MDS file from the combined data set and the IBS values.
12. Creates a population file for plotting purposes.
13. Plots the MDS values.
14. Finds the outliers of a given reference population (either CEU, YRI or JPT-CHB).
    (a) Reads the population file.
    (b) Reads the MDS values.
    (c) Computes the three reference population clusters' center.
    (d) Computes three clusters according to the reference population clusters' centers, and finds the outliers of a given reference population.
    (e) Writes the outliers in a file.

5.11.3 Output Files

The output files of each of the steps described above are as follow (note that the output prefix shown is the one by default [*i.e* `ethnicity`]):

1. Three files are created:
   - `ethnicity.ref_snp_to_extract`: a list of markers to extract from the reference panels.
   - `ethnicity.source_snp_to_extract`: a list of markers to extract from the source panel.
   - `ethnicity.update_names`: the updated names of the marker in the reference panels, so that they match with the names in the source panel.
2. Four sets of PLINK's binary files are created:
   - `ethnicity.reference_panel.CEU`: the data set containing the extracted markers from the CEU reference population.
   - `ethnicity.reference_panel.YRI`: the data set containing the extracted markers from the YRI reference population.
   - `ethnicity.reference_panel.JPT-CHB`: the data set containing the extracted markers from the JPG-CHB reference population.
   - `ethnicity.source_panel.ALL`: the data set containing the extracted markers from the source population.
3. One required file and one set of PLINK's binary files are created:
   - `ethnicity.reference_panel.ALL.files_to_merge`: the file required by Plink to merge more than two data sets together.
   - `ethnicity.reference_panel.ALL`: the data set containing the merged data sets of the three reference population.
4. One set of PLINK's binary files is created:
   - `ethnicity.reference_panel.ALL.rename`: the data set after markers have been renamed in the reference panels.
5. Two sets of PLINK's result files are created:

- `ethnicity.reference_panel.ALL.rename.frequency`: the frequencies of the markers in the reference panels.
- `ethnicity.source_panel.ALL.frequency`: the frequencies of the markers in the source panels.

6. Two files are created:
   - `ethnicity.snp_to_flip_in_reference`: the list of markers to flip in the reference panels.
   - `ethnicity.snp_to_remove`: the list of markers to remove because they are not comparable to the markers in the source panel, even after trying to flip them.
7. Two sets of PLINK's binary files are created:
   - `ethnicity.reference_panel.ALL.rename.cleaned`: the data set after the markers found in the previous step are excluded from the reference panels.
   - `ethnicity.source_panel.ALL.cleaned`: the data set after the markers found in the previous step are excluded from the source panel.
8. One set of PLINK's binary files is created:
   - `ethnicity.reference_panel.ALL.rename.cleaned.flipped`: the data set after markers from the reference panels were flipped so that they become comparable with the source panel.
9. One required file and one set of PLINK's binary files are created:
   - `ethnicity.final_dataset_for_genome.files_to_merge`: the file required by Plink to merge more than two data sets together.
   - `ethnicity.final_dataset_for_genome`: the data set containing the merged reference and source panels.
10. Multiple files are created after this step.
    - `ethnicity.ibs` : for more information about those files, see Section *Related Samples Module*.
11. One set of PLINK's result files is created:
    - `ethnicity.mds`: files containing the MDS values.
12. One file is created:
    - `ethnicity.population_file`: the population file required for MDS value plotting.
13. One file is created:
    - `ethnicity.mds.png`: the plot of the MDS values (see Figure *Initial MDS plot*).
14. Four files are created:
    - `ethnicity.before.png`: the MDS values before outliers detection (see Figure *MDS plot before outlier detection*).
    - `ethnicity.after.png`: the MDS values after outliers detection for each of the three reference populations. The shaded points are the outliers (see Figure *MDS plot after outlier detection*).
    - `ethnicity.outliers.png`: the MDS values after outliers detection for the selected reference population (default is CEU) (see Figure *Ethnic outliers*).
    - `ethnicity.outliers`: the list of outliers (excluding the reference populations).
    - `ethnicity.population_file_outliers`: a population file containing the outliers (to help creating a new MDS plot using `PlinkUtils.plot_MDS_standalone`).

5.11.4 The Plots

Multiple plots are created by this module. The first one (Figure *Initial MDS plot*) is the MDS values right after they are computed by Plink. There is one color per reference populations (CEU in blue, YRI in green and JPT-CHB in purple). The source population is represented as red crosses.

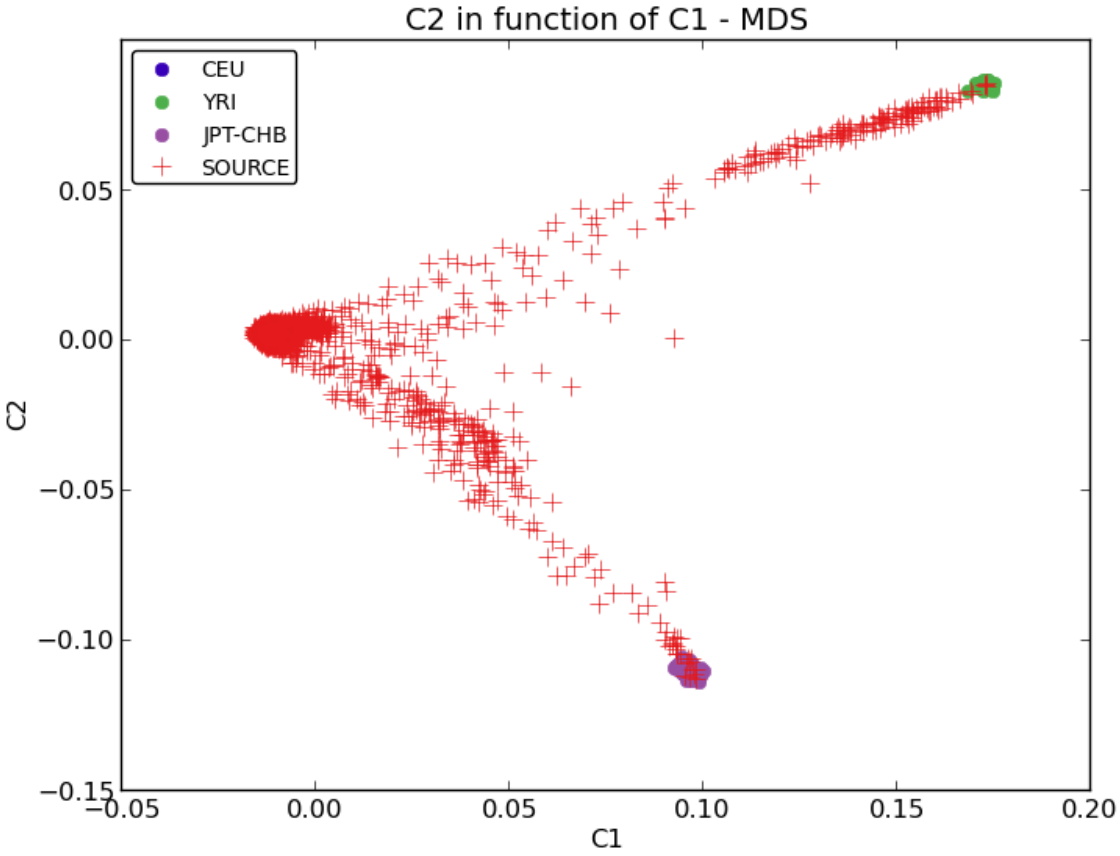


Figure 5.7: Initial MDS plot

The second one (Figure *MDS plot before outlier detection*) is the MDS values before outlier detection. Points in red, green and blue represent the individuals part of the CEU, YRI and JPT-CHB clusters, respectively. The yellow points represent the center of each of the cluster, when only considering the three reference panels.

The third plot (Figure *MDS plot after outlier detection*) is the MDS values after outlier detection. Points in red, green and blue represent the individuals part of the CEU, YRI and JPT-CHB clusters, respectively. Outliers are found for each of the three reference populations and they are represented with the same, but lighter color. Once again, the yellow points represent the center of each of the cluster, when only considering the three reference panels.

The last plot (Figure *Ethnic outliers*) shows the outliers of the selected reference population (CEU by default). Red, green and blue represent the CEU, YRI and JPT-CHB samples, respectively. Orange represents the individuals from the source panel who are part of the selected reference population. Gray represents the outliers of the selected reference population.

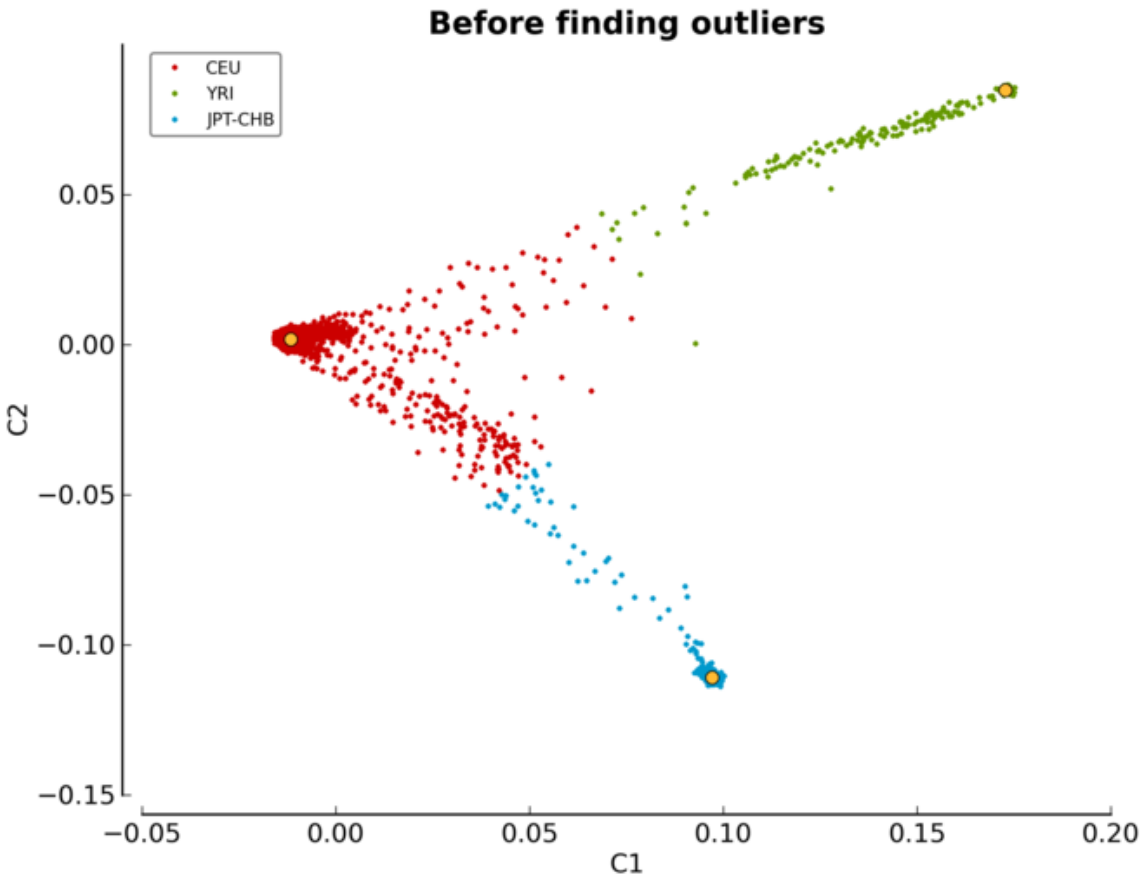


Figure 5.8: MDS plot before outlier detection

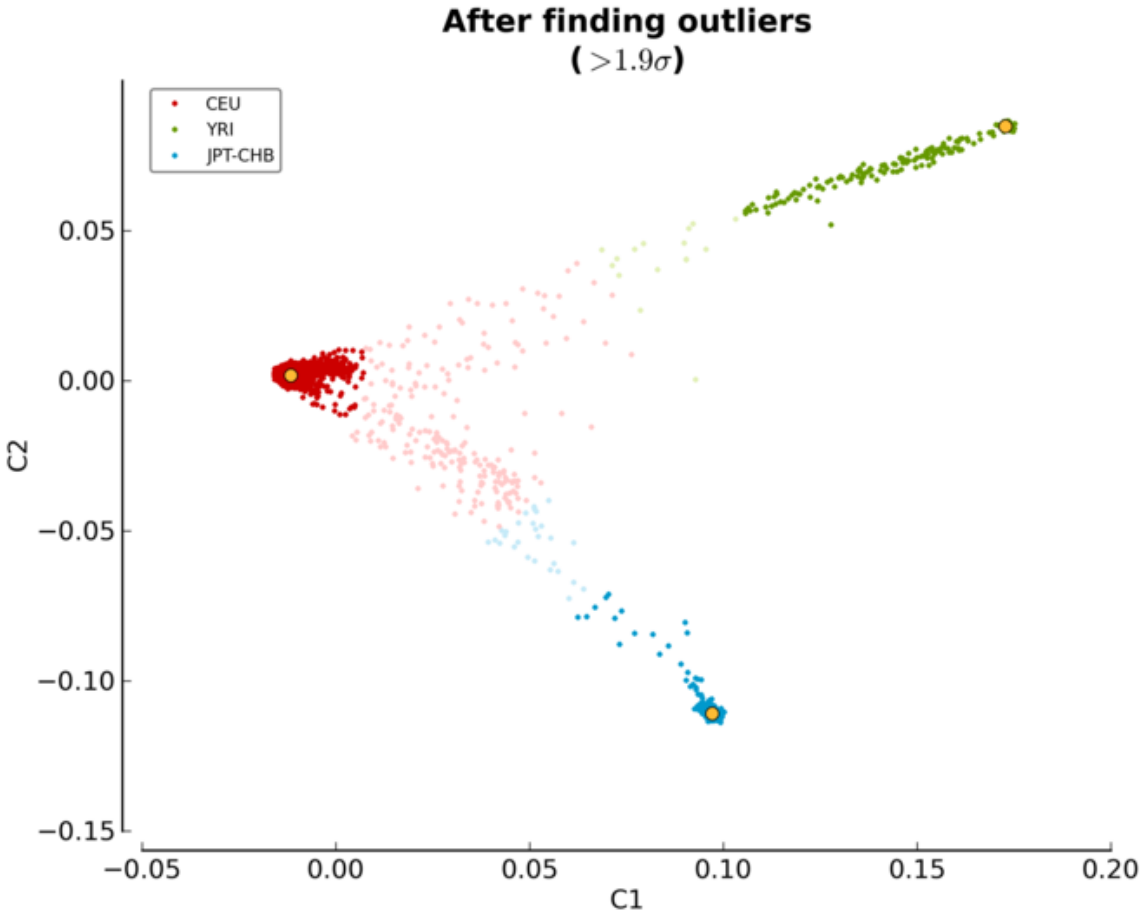


Figure 5.9: MDS plot after outlier detection

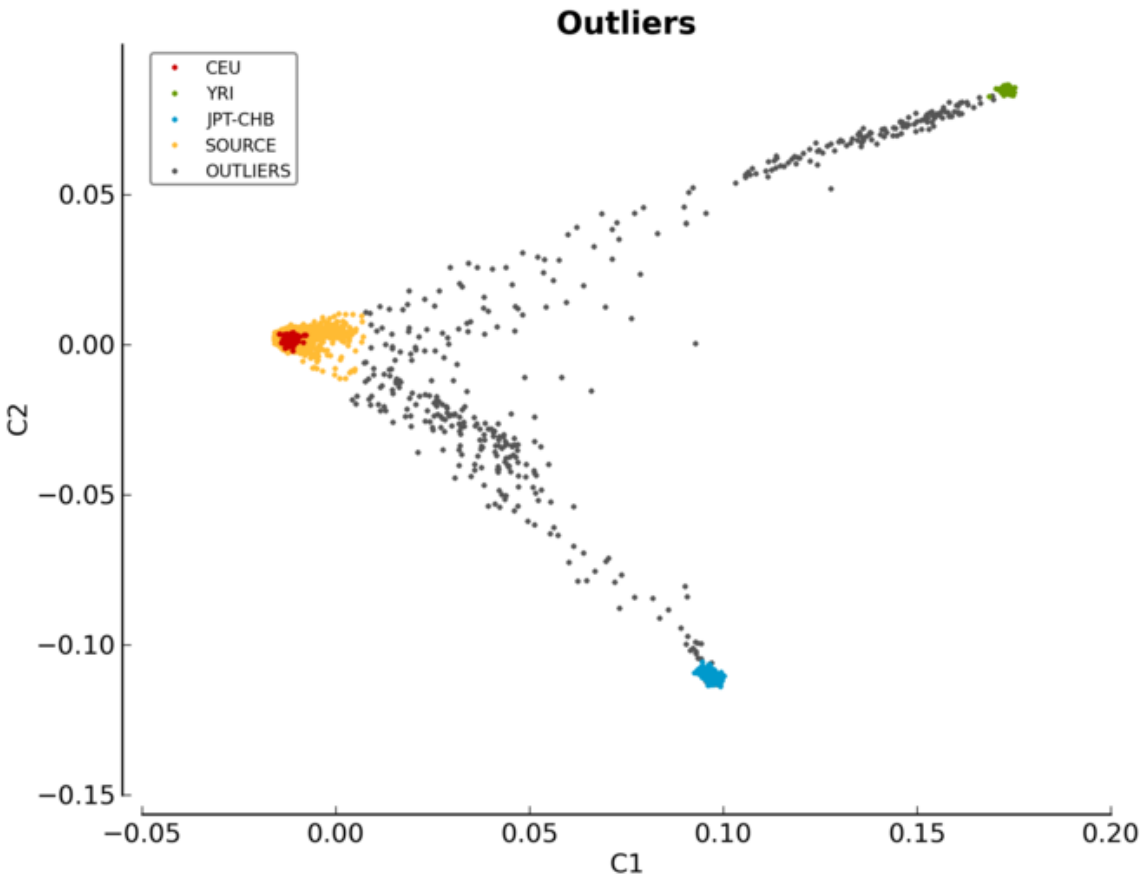


Figure 5.10: Ethnic outliers

Modifying The Outlier Plot

If you want to manually modify the above figures, have a look at the `PlinkUtils.plot_MDS_standalone` module. Here is the usage of this script:

```
$ pyGenClean_plot_MDS --help
usage: pyGenClean_plot_MDS [-h] --file FILE --population-file FORMAT
                           [--population-order STRING]
                           [--population-colors STRING]
                           [--population-sizes STRING]
                           [--population-markers STRING]
                           [--population-alpha STRING] [--format FORMAT]
                           [--title STRING] [--xaxis STRING] [--xlabel STRING]
                           [--yaxis STRING] [--ylabel STRING]
                           [--legend-position STRING] [--legend-size INT]
                           [--legend-ncol INT] [--legend-alpha FLOAT]
                           [--title-fontsize INT] [--label-fontsize INT]
                           [--axis-fontsize INT] [--adjust-left FLOAT]
                           [--adjust-right FLOAT] [--adjust-top FLOAT]
                           [--adjust-bottom FLOAT] [--out FILE]

Creates a MDS plot

optional arguments:
  -h, --help            show this help message and exit

Input File:
  --file FILE           The MBS file.
  --population-file FORMAT
                        A file containing population information. There must
                        be three columns: famID, indID and population
                        information.

Population Properties:
  --population-order STRING
                        The order to print the different populations.
                        [default: CEU,YRI,JPT-CHB,SOURCE,OUTLIER]
  --population-colors STRING
                        The population point color in the plot [default:
                        377eb8,4daf4a,984ea3,e41a1c,ff7f00]
  --population-sizes STRING
                        The population point size in the plot. [default:
                        12,12,12,8,3]
  --population-markers STRING
                        The population point marker in the plot. [default:
                        .,.,.,+,D]
  --population-alpha STRING
                        The population alpha value in the plot. [default:
                        1.0,1.0,1.0,1.0,1.0]

Graphical Properties:
  --format FORMAT       The output file format (png, ps, pdf, or X11 formats
                        are available). [default: png]
  --title STRING        The title of the MDS plot. [default: C2 in function of
                        C1 - MDS]
  --xaxis STRING        The component to print on the X axis. [default: C1]
  --xlabel STRING       The label of the X axis. [default: C1]
  --yaxis STRING        The component to print on the Y axis. [default: C2]
```

```
  --ylabel STRING       The label of the Y axis. [default: C2]
  --legend-position STRING
                        The position of the legend. [default: best]
  --legend-size INT     The size of the legend. [default: 10]
  --legend-ncol INT     The number of column for the legend. [default: 1]
  --legend-alpha FLOAT  The alpha value of the legend frame. [default: 1.0]
  --title-fontsize INT  The font size of the title. [default: 15]
  --label-fontsize INT  The font size of the X and Y labels. [default: 12]
  --axis-fontsize INT   The font size of the X and Y axis. [Default: 12]
  --adjust-left FLOAT   Adjust the left margin. [Default: 0.12]
  --adjust-right FLOAT  Adjust the right margin. [Default: 0.90]
  --adjust-top FLOAT    Adjust the top margin. [Default: 0.90]
  --adjust-bottom FLOAT
                        Adjust the bottom margin. [Default: 0.10]

Output File:
  --out FILE            The prefix of the output files. [default: mds]
```

And here is an example of usage (for a MDS and a population file named `ethnicity.mds.mds` and `ethnicity.population_file_outliers`, respectively), producing the Figure *Ethnic outliers modified*.

```
$ pyGenClean_plot_MDS \
>     --file ethnicity.mds.mds \
>     --population-file ethnicity.population_file_outliers \
>     --population-order SOURCE,CEU,YRI,JPT-CHB,OUTLIER \
>     --population-colors e41a1c,377eb8,4daf4a,984ea3,000000 \
>     --population-markers .,.,.,.,+ \
>     --population-sizes 8,8,8,8,8 \
>     --axis-fontsize 18 \
>     --label-fontsize 18 \
>     --title-fontsize 24 \
>     --adjust-bottom 0.11 \
>     --adjust-left 0.15 \
>     --adjust-right 0.96 \
>     --legend-size 14 \
>     --legend-position lower-right
```

5.11.5 Finding Outliers

If the multiplier of the cluster standard deviation was too stringent (or not stringent enough), there is no need to run the module from the start. A standalone script was created for this exact purpose, and it will find the outliers using the `MDS` and population file previously created. Just modify the `--multiplier` option and restart the analysis (which takes about a couple of seconds).

```
$ pyGenClean_find_outliers --help
usage: pyGenClean_find_outliers [-h] --mds FILE --population-file FILE
                                [--outliers-of POP] [--multiplier FLOAT]
                                [--xaxis COMPONENT] [--yaxis COMPONENT]
                                [--format FORMAT] [--out FILE]

Finds outliers in SOURCE from CEU samples.

optional arguments:
  -h, --help            show this help message and exit

Input File:
  --mds FILE            The MDS file from Plink
```

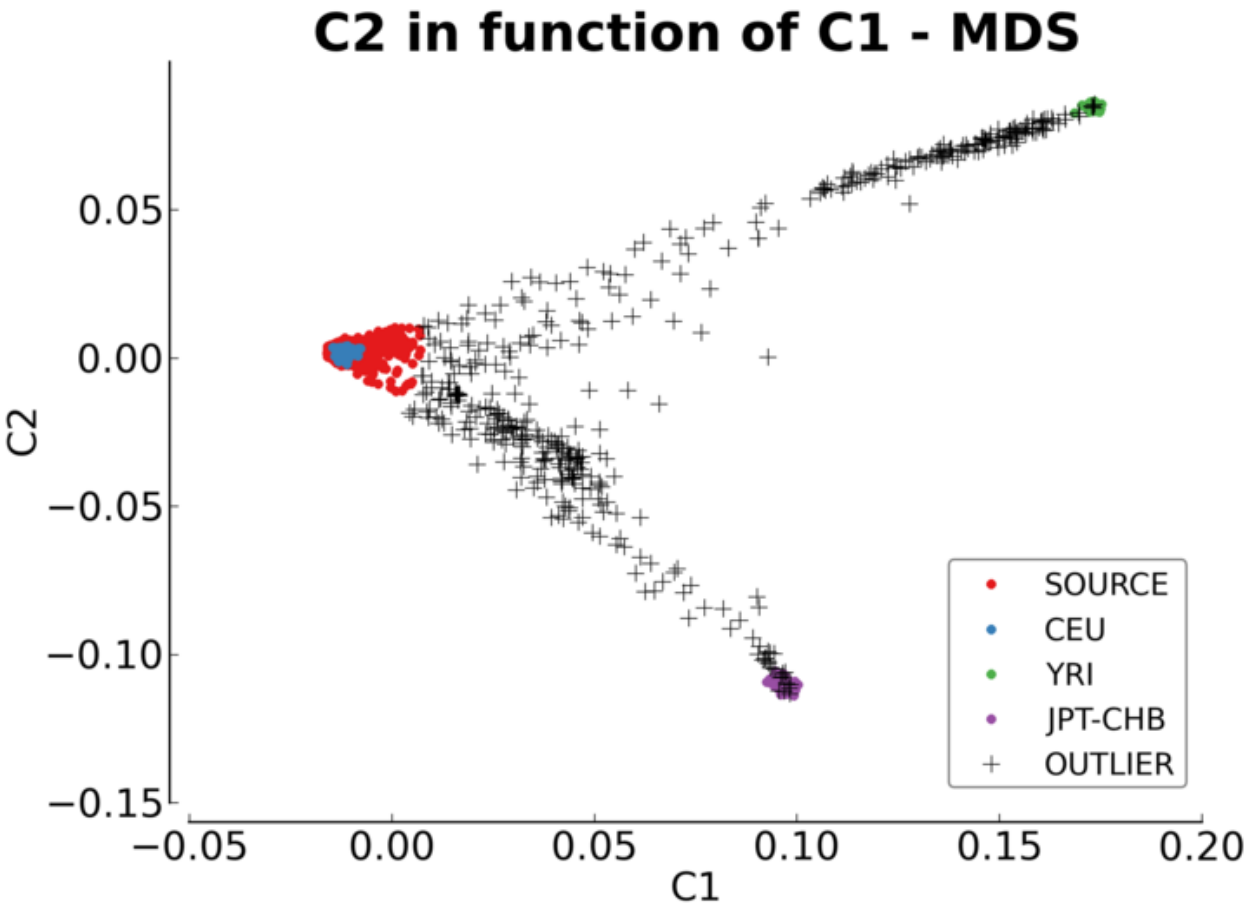


Figure 5.11: Ethnic outliers modified

```
  --population-file FILE
                        A population file containing the following columns
                        (without a header): FID, IID and POP. POP should be
                        one of 'CEU', 'JPT-CHB', 'YRI' and SOURCE.

Options:
  --outliers-of POP     Finds the outliers of this population. [default: CEU]
  --multiplier FLOAT    To find the outliers, we look for more than x times
                        the cluster standard deviation. [default: 1.9]
  --xaxis COMPONENT     The component to use for the X axis. [default: C1]
  --yaxis COMPONENT     The component to use for the Y axis. [default: C2]
  --format FORMAT       The output file format (png, ps, or pdf formats are
                        available). [default: png]

Output File:
  --out FILE            The prefix of the output files. [default: ethnicity]
```

5.11.6 The Algorithm

For more information about the actual algorithms and source codes (the `Step10.check_ethnicity`, the `Step10.find_outliers` and the `PlinkUtils.plot_MDS_standalone` modules), refer to the following sections.

Step10.check_ethnicity

exception `Step10.check_ethnicity.`**`ProgramError`**(*msg*)
An `Exception` raised in case of a problem.

Parameters **msg** (*string*) – the message to print to the user before exiting.

`Step10.check_ethnicity.`**`allFileExists`**(*fileList*)
Check that all file exists.

Parameters **fileList** (*list of strings*) – the list of file to check.

Check if all the files in `fileList` exists.

`Step10.check_ethnicity.`**`checkArgs`**(*args*)
Checks the arguments and options.

Parameters **args** (*argparse.Namespace*) – an object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the `ProgramError` class, a message is printed to the `sys.stderr` and the program exists with code 1.

`Step10.check_ethnicity.`**`combinePlinkBinaryFiles`**(*prefixes*, *outPrefix*)
Combine Plink binary files.

Parameters

- **prefixes** (*list of strings*) – a list of the prefix of the files that need to be combined.
- **outPrefix** (*string*) – the prefix of the output file (the combined file).

It uses Plink to merge a list of binary files (which is a list of prefixes (strings)), and create the final data set which as `outPrefix` as the prefix.

Step10.check_ethnicity.**computeFrequency**(*prefix*, *outPrefix*)
Compute the frequency using Plink.

Parameters

- **prefix** (*string*) – the prefix of the file binary file for which we need to compute frequencies.
- **outPrefix** (*string*) – the prefix of the output files.

Uses Plink to compute the frequency of all the markers in the `prefix` binary file.

Step10.check_ethnicity.**createMDSFile**(*nb_components*, *inPrefix*, *outPrefix*, *genomeFileName*)
Creates a MDS file using Plink.

Parameters

- **nb_components** (*int*) – the number of component.
- **inPrefix** (*string*) – the prefix of the input file.
- **outPrefix** (*string*) – the prefix of the output file.
- **genomeFileName** (*string*) – the name of the `genome` file.

Using Plink, computes the MDS values for each individual using the `inPrefix`, `genomeFileName` and the number of components. The results are save using the `outPrefix` prefix.

Step10.check_ethnicity.**createPopulationFile**(*inputFiles*, *labels*, *outputFileName*)
Creates a population file.

Parameters

- **inputFiles** (*list of strings*) – the list of input files.
- **labels** (*list of strings*) – the list of labels (corresponding to the input files).
- **outputFileName** (*string*) – the name of the output file.

The `inputFiles` is in reality a list of `tfam` files composed of samples. For each of those `tfam` files, there is a label associated with it (representing the name of the population).

The output file consists of one row per sample, with the following three columns: the family ID, the individual ID and the population of each sample.

Step10.check_ethnicity.**excludeSNPs**(*inPrefix*, *outPrefix*, *exclusionFileName*)
Exclude some SNPs using Plink.

Parameters

- **inPrefix** (*string*) – the prefix of the input file.
- **outPrefix** (*string*) – the prefix of the output file.
- **exclusionFileName** (*string*) – the name of the file containing the markers to be excluded.

Using Plink, exclude a list of markers from `inPrefix`, and saves the results in `outPrefix`. The list of markers are in `exclusionFileName`.

Step10.check_ethnicity.**extractSNPs**(*snpToExtractFileName*, *referencePrefixes*, *popNames*, *outPrefix*, *runSGE*)
Extract a list of SNPs using Plink.

Parameters

- **snpToExtractFileName** (*string*) – the name of the file which contains the markers to extract from the original data set.

- **referencePrefixes** (*list of string*) – a list containing the three reference population prefixes (the original data sets).
- **popNames** (*list of string*) – a list containing the three reference population names.
- **outPrefix** (*string*) – the prefix of the output file.
- **runSGE** (*boolean*) – Whether using SGE or not.

Using Plink, extract a set of markers from a list of prefixes.

`Step10.check_ethnicity.`**`findFlippedSNPs`**(*frqFile1*, *frqFile2*, *outPrefix*)

Find flipped SNPs and flip them in the data.

Parameters

- **frqFile1** (*string*) – the name of the first frequency file.
- **frqFile2** (*string*) – the name of the second frequency file.
- **outPrefix** (*string*) – the prefix of the output files.

By reading two frequency files (`frqFile1` and `frqFile2`), it finds a list of markers that need to be flipped so that the first file becomes comparable with the second one. Also finds marker that need to be removed.

A marker needs to be flipped in one of the two data set if the two markers are not comparable (same minor allele), but become comparable if we flip one of them.

A marker will be removed if it is all homozygous in at least one data set. It will also be removed if it's impossible to determine the phase of the marker (*e.g.* if the two alleles are `A` and `T` or `C` and `G`).

`Step10.check_ethnicity.`**`findOverlappingSNPsWithReference`**(*prefix*, *referencePrefixes*, *outPrefix*)

Find the overlapping SNPs in 4 different data sets.

Parameters

- **prefix** (*string*) – the prefix of all the files.
- **referencePrefixes** (*list of strings*) – the prefix of the reference population files.
- **outPrefix** (*string*) – the prefix of the output files.

It starts by reading the `bim` file of the source data set (`prefix.bim`). It finds all the markers (excluding the duplicated ones). Then it reads all of the reference population `bim` files (`referencePrefixes.bim`) and find all the markers that were found in the source data set.

It creates three output files:

- `outPrefix.ref_snp_to_extract`: the name of the markers that needs to be extracted from the three reference panels.
- `outPrefix.source_snp_to_extract`: the name of the markers that needs to be extracted from the source panel.
- `outPrefix.update_names`: a file (readable by Plink) that will help in changing the names of the selected markers in the reference panels, so that they become comparable with the source panel.

`Step10.check_ethnicity.`**`find_the_outliers`**(*mds_file_name*, *population_file_name*, *ref_pop_name*, *multiplier*, *out_prefix*)

Finds the outliers of a given population.

Parameters

- **mds_file_name** (*string*) – the name of the `mds` file.
- **population_file_name** (*string*) – the name of the population file.

- **ref_pop_name** (*string*) – the name of the reference population for which to find outliers from.
- **multiplier** (*float*) – the multiplier of the cluster standard deviation to modify the strictness of the outlier removal procedure.
- **out_prefix** (*string*) – the prefix of the output file.

Uses the `Step10.find_outliers` modules to find outliers. It requires the `mds` file created by `createMDSFile()` and the population file created by `createPopulationFile()`.

`Step10.check_ethnicity.`**`flipSNPs`**(*inPrefix*, *outPrefix*, *flipFileName*)

Flip SNPs using Plink.

Parameters

- **inPrefix** (*string*) – the prefix of the input file.
- **outPrefix** (*string*) – the prefix of the output file.
- **flipFileName** (*string*) – the name of the file containing the markers to flip.

Using Plink, flip a set of markers in `inPrefix`, and saves the results in `outPrefix`. The list of markers to be flipped is in `flipFileName`.

`Step10.check_ethnicity.`**`main`**(*argString=None*)

The main function.

Parameters **argString** (*list of strings*) – the options.

These are the steps of this module:

1. Prints the options.
2. Finds the overlapping markers between the three reference panels and the source panel (`findOverlappingSNPsWithReference()`).
3. Extract the required markers from all the data sets (`extractSNPs()`).
4. Combines the three reference panels together (`combinePlinkBinaryFiles()`).
5. Renames the reference panel's marker names to that they are the same as the source panel (`renameSNPs()`).
6. Compute the frequency of all the markers from the reference and the source panels (`computeFrequency()`).
7. Finds the markers to flip in the reference panel (when compared to the source panel) (`findFlippedSNPs()`).
8. Excludes the unflippable markers from the reference and the source panels (`excludeSNPs()`).
9. Flips the markers that need flipping in ther reference panel (`flipSNPs()`).
10. Combines the reference and the source panels (`combinePlinkBinaryFiles()`).
11. Runs part of `Step9.find_related_samples` on the combined data set (`runTheStep9()`).
12. Creates the `mds` file from the combined data set and the result of previous step (`createMDSFile()`).
13. Creates the population file (`createPopulationFile()`).
14. Plots the `mds` values (`plotMDS()`).
15. Finds the outliers of a given reference population (`find_the_outliers()`).

`Step10.check_ethnicity.`**`parseArgs`**(*argString=None*)

Parses the command line options and arguments.

Parameters **argString** (*list of string*) – the options.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--bfile`	string	The input file prefix (Plink binary file).
`--ceu-bfile`	string	The input file prefix for the CEU population (Plink binary file).
`--yri-bfile`	string	The input file prefix for the YRI population (Plink binary file).
`--jpt-chb-bfile`	string	The input file prefix for the JPT-CHB population (Plink binary file).
`--min-nb-snp`	int	The minimum number of markers needed to compute IBS.
`--indep-pairwise`	string	Three numbers: window size, window shift and the r2 threshold.
`--maf`	string	Restrict to SNPs with MAF >= threshold.
`--sge`	bool	Use SGE for parallelization.
`--line-per-file-for-sge`	int	The number of line per file for SGE task array.
`--nb-components`	int	The number of component to compute.
`--outliers-of`	string	Finds the ouliers of this population.
`--multiplier`	float	To find the outliers, we look for more than x times the cluster standard deviation.
`--xaxis`	string	The component to use for the X axis.
`--yaxis`	string	The component to use for the Y axis.
`--format`	string	The output file format.
`--title`	string	The title of the MDS plot.
`--xlabel`	string	The label of the X axis.
`--ylabel`	string	The label of the Y axis.
`--out`	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see `checkArgs()`).

`Step10.check_ethnicity.`**`plotMDS`**(*inputFileName*, *outPrefix*, *populationFileName*, *options*)

Plots the MDS value.

Parameters

- **inputFileName** (*string*) – the name of the `mds` file.
- **outPrefix** (*string*) – the prefix of the output files.
- **populationFileName** (*string*) – the name of the population file.
- **options** (*argparse.Namespace*) – the options

Plots the `mds` value according to the `inputFileName` file (`mds`) and the `populationFileName` (the population file).

`Step10.check_ethnicity.`**`renameSNPs`**(*inPrefix*, *updateFileName*, *outPrefix*)

Updates the name of the SNPs using Plink.

Parameters

- **inPrefix** (*string*) – the prefix of the input file.
- **updateFileName** (*string*) – the name of the file containing the updated marker names.
- **outPrefix** (*string*) – the prefix of the output file.

Using Plink, changes the name of the markers in `inPrefix` using `updateFileName`. It saves the results in `outPrefix`.

Step10.check_ethnicity.**runCommand**(*command*)
Run a command.

Parameters **command** (*list of strings*) – the command to run.

Tries to run a command. If it fails, raise a ProgramError. This function uses the subprocess module.

Warning: The variable command should be a list fo strings (no other type).

Step10.check_ethnicity.**runTheStep9**(*inputPrefix*, *outPrefix*, *options*)
Run the Step9 of the data clean up.

Parameters

- **inputPrefix** (*string*) – the prefix of the input file.
- **outPrefix** (*string*) – the prefix of the output file.
- **options** (*argparse.Namespace*) – the options

Returns the prefix of the new bfile.

Runs Step9.find_related_samples using the inputPrefix files and options options, and saves the results using the outPrefix prefix.

Step10.find_outliers

exception Step10.find_outliers.**ProgramError**(*msg*)
An Exception raised in case of a problem.

Parameters **msg** (*string*) – the message to print to the user before exiting.

Step10.find_outliers.**add_custom_options**(*parser*)
Adds custom options to a parser.

Parameters **parser** (*argparse.ArgumentParser*) – the parser to which to add options.

Step10.find_outliers.**checkArgs**(*args*)
Checks the arguments and options.

Parameters **args** (*argparse.Namespace*) – a argparse.Namespace object containing the options of the program.

Returns True if everything was OK.

If there is a problem with an option, an exception is raised using the ProgramError class, a message is printed to the sys.stderr and the program exists with code 1.

Step10.find_outliers.**find_outliers**(*mds*, *centers*, *center_info*, *ref_pop*, *options*)
Finds the outliers for a given population.

Parameters

- **mds** (*numpy.recarray*) – the mds information about each samples.
- **centers** (*numpy.array*) – the centers of the three reference population clusters.
- **center_info** (*dict*) – the label of the three reference population clusters.
- **ref_pop** (*string*) – the reference population for which we need the outliers from.
- **options** (*argparse.Namespace*) – the options

Returns a set of outliers from the ref_pop population.

Perform a `KMeans` classification using the three centers from the three reference population cluster.

Samples are outliers of the required reference population (`ref_pop`) if:

- the sample is part of another reference population cluster;
- the sample is an outlier of the desired reference population (`ref_pop`).

A sample is an outlier of a given cluster C_j if the distance between this sample and the center of the cluster C_j (O_j) is bigger than a constant times the cluster's standard deviation σ_j.

$$\sigma_j = \sqrt{\frac{\sum d(s_i, O_j)^2}{||C_j|| - 1}}$$

where $||C_j||$ is the number of samples in the cluster C_j, and $d(s_i, O_j)$ is the distance between the sample s_i and the center O_j of the cluster C_j.

$$d(s_i, O_j) = \sqrt{(x_{O_j} - x_{s_i})^2 + (y_{O_j} - y_{s_i})^2}$$

Using a constant equals to one ensure we remove 100% of the outliers from the cluster. Using a constant of 1.6 or 1.9 ensures we remove 99% and 95% of outliers, respectively (an error rate of 1% and 5%, respectively).

`Step10.find_outliers.`**`find_ref_centers`**(*mds*)

Finds the center of the three reference clusters.

Parameters **mds** (*numpy.recarray*) – the `mds` information about each samples.

Returns a tuple with a `numpy.array` containing the centers of the three reference population cluster as first element, and a `dict` containing the label of each of the three reference population clusters.

First, we extract the `mds` values of each of the three reference populations. The, we compute the center of each of those clusters by computing the means.

$$\text{Cluster}_{\text{pop}} = \left(\frac{\sum_{i=1}^{n} x_i}{n}, \frac{\sum_{i=1}^{n} y_i}{n} \right)$$

`Step10.find_outliers.`**`main`**(*argString=None*)

The main function.

Parameters **argString** (*list of strings*) – the options.

These are the steps of the modules:

1. Prints the options.
2. Reads the population file (`read_population_file()`).
3. Reads the `mds` file (`read_mds_file()`).
4. Computes the three reference population clusters' centers (`find_ref_centers()`).
5. Computes three clusters according to the reference population clusters' centers, and finds the outliers of a given reference population (`find_outliers()`). This steps also produce three different plots.
6. Writes outliers in a file (`prefix.outliers`).

`Step10.find_outliers.`**`parseArgs`**(*argString=None*)

Parses the command line options and arguments.

Parameters **argString** (*list of string*) – the options.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--mds`	string	The MDS file from Plink.
`--population-file`	string	A population file from `Step10.check_ethnicity` module.
`--format`	string	The output file format (png, ps, or pdf.
`--out`	string	The prefix of the output files.
`--outliers-of`	string	Finds the outliers of this population.
`--multiplier`	float	To find the outliers, we look for more than x times the cluster standard deviation.
`--xaxis`	string	The component to use for the X axis.
`--yaxis`	string	The component to use for the Y axis.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see `checkArgs()`).

`Step10.find_outliers.`**`read_mds_file`**(*file_name*, *c1*, *c2*, *pops*)

Reads a MDS file.

Parameters

- **file_name** (*string*) – the name of the `mds` file.
- **c1** (*string*) – the first component to read (x axis).
- **c2** (*string*) – the second component to read (y axis).
- **pops** (*dict*) – the population of each sample.

Returns a `numpy.recarray` (one sample per line) with the information about the family ID, the individual ID, the first component to extract, the second component to extract and the population.

The `mds` file is the result of Plink (as produced by the `Step10.check_ethnicity` module).

`Step10.find_outliers.`**`read_population_file`**(*file_name*)

Reads the population file.

Parameters **file_name** (*string*) – the name of the population file.

Returns a `dict` containing the population for each of the samples.

The population file should contain three columns:

1. The family ID.
2. The individual ID.
3. The population of the file (one of `CEU`, `YRI`, `JPT-CHB` or `SOURCE`).

The outliers are from the `SOURCE` population, when compared to one of the three reference population (`CEU`, `YRI` or `JPT-CHB`).

PlinkUtils.plot_MDS_standalone

exception `PlinkUtils.plot_MDS_standalone.`**`ProgramError`**(*msg*)

An `Exception` raised in case of a problem.

Parameters **msg** (*string*) – the message to print to the user before exiting.

PlinkUtils.plot_MDS_standalone.**checkArgs**(*args*)
Checks the arguments and options.

Parameters **args** (*argparse.Namespace*) – an object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the `ProgramError` class, a message is printed to the `sys.stderr` and the program exists with code 1.

PlinkUtils.plot_MDS_standalone.**extractData**(*fileName*, *populations*, *population_order*, *xaxis*, *yaxis*)
Extract the C1 and C2 columns for plotting.

Parameters

- **fileName** (*string*) – the name of the MDS file.
- **populations** (*dict*) – the population of each sample in the MDS file.
- **population_order** (*list of string*) – the required population order.
- **xaxis** (*string*) – the component to print as the X axis.
- **yaxis** (*string*) – the component to print as the Y axis.

Returns the MDS data with information about the population of each sample. The first element of the returned tuple is a tuple. The last element of the returned tuple is the list of the populations (the order is the same as in the first element). The first element of the first tuple is the C1 data, and the last element is the C2 data.

Note: If a sample in the MDS file is not in the population file, it is skip.

PlinkUtils.plot_MDS_standalone.**main**()
The main function of the module.

These are the steps:

1. Reads the population file (`readPopulations()`).
2. Extracts the MDS values (`extractData()`).
3. Plots the MDS values (`plotMDS()`).

PlinkUtils.plot_MDS_standalone.**parseArgs**()
Parses the command line options and arguments.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--file`	string	The MBS file.
`--population-file`	string	A file containing population information.
`--population-order`	string	The order to print the different populations.
`--population-colors`	string	The population point color in the plot.
`--population-sizes`	string	The population point size in the plot.
`--population-markers`	string	The population point marker in the plot.
`--population-alpha`	string	The population alpha value in the plot.
`--format`	string	The output file format.
`--title`	string	The title of the MDS plot.
`--xaxis`	string	The component to print on the X axis.
`--xlabel`	string	The label of the X axis.
`--yaxis`	string	The component to print on the Y axis.
`--ylabel`	string	The label of the Y axis.
`--legend-position`	string	The position of the legend.
`--legend-size`	int	The size of the legend text.
`--legend-ncol`	int	The number of columns for the legend.
`--legend-alpha`	float	The alpha value of the legend.
`--title-fontsize`	int	The font size of the title.
`--label-fontsize`	int	The font size of the X and Y labels.
`--axis-fontsize`	int	The font size of the X and Y axis.
`--adjust-left`	float	Adjust the left margin.
`--adjust-right`	float	Adjust the right margin.
`--adjust-top`	float	Adjust the top margin.
`--adjust-bottom`	float	Adjust the bottom margin.
`--out`	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see `checkArgs()`).

`PlinkUtils.plot_MDS_standalone.`**`plotMDS`**(*data*, *theOrders*, *theLabels*, *theColors*, *theAlphas*, *theSizes*, *theMarkers*, *options*)

Plot the MDS data.

Parameters

- **data** (*list of numpy.array*) – the data to plot (MDS values).
- **theOrders** (*list of int*) – the order of the populations to plot.
- **theLabels** (*list of string*) – the names of the populations to plot.
- **theColors** (*list of string*) – the colors of the populations to plot.
- **theAlphas** (*list of float*) – the alpha value for the populations to plot.
- **theSizes** (*list of int*) – the sizes of the markers for each population to plot.
- **theMarkers** (*list of string*) – the type of marker for each population to plot.
- **options** (*argparse.Namespace*) – the options.

`PlinkUtils.plot_MDS_standalone.`**`readPopulations`**(*inputFileName*, *requiredPopulation*)

Reads a population file.

Parameters

- **inputFileName** (*string*) – the name of the population file.
- **requiredPopulation** (*list of string*) – the required population.

Returns a `dict` containing the population of each samples.

5.12 Minor Allele Frequency of Zero Module

The usage of the standalone module is shown below:

```
$ pyGenClean_flag_maf_zero --help
usage: pyGenClean_flag_maf_zero [-h] --bfile FILE [--out FILE]

Flag SNPs with MAF of 0.

optional arguments:
  -h, --help    show this help message and exit

Input File:
  --bfile FILE  The input file prefix (will find the plink binary files by
                appending the prefix to the .bim, .bed and .fam files,
                respectively.

Output File:
  --out FILE    The prefix of the output files. [default: flag_maf_0]
```

5.12.1 Input FIles

This module uses PLINK's binary file format (`bed`, `bim` and `fam` files) for the source data set (the data of interest).

5.12.2 Procedure

Here are the steps performed by the module:

1. Computes the frequencies using Plink.
2. Finds markers with a MAF of zero.

5.12.3 Output Files

The output files of each of the steps described above are as follow (note that the output prefix shown is the one by default [*i.e.* `flag_maf_0`]):

1. One file and one set of PLINK's result file:
    - `flag_maf_0`: the frequency of each marker in the source dataset.
    - `flag_maf_0.list`: the list of markers with a minor allele frequency of zero.

5.12.4 The Algorithm

For more information about the actual algorithms and source codes (the `Step11.flag_maf_zero` module), refer to the following sections.

Step11.flag_maf_zero

exception Step11.flag_maf_zero.**ProgramError**(*msg*)
An Exception raised in case of a problem.

Parameters **msg** (*string*) – the message to print to the user before exiting.

Step11.flag_maf_zero.**checkArgs**(*args*)
Checks the arguments and options.

Parameters **args** (*argparse.Namespace*) – a argparse.Namespace object containing the options of the program.

Returns True if everything was OK.

If there is a problem with an option, an exception is raised using the ProgramError class, a message is printed to the sys.stderr and the program exists with code 1.

Step11.flag_maf_zero.**computeFrequency**(*options*)
Compute the frequency of the SNPs.

Parameters **options** (*argparse.Namespace*) – the options.

Step11.flag_maf_zero.**findSnpWithMaf0**(*freqFileName*, *prefix*)
Finds SNPs with MAF of 0 and put them in a file.

Parameters

- **freqFileName** (*string*) – the name of the frequency file.
- **prefix** (*string*) – the prefix of all the files.

Reads a frequency file from Plink, and find markers with a minor allele frequency of zero.

Step11.flag_maf_zero.**main**(*argString=None*)
The main function.

Parameters **argString** (*list of strings*) – the options.

These are the steps:

1. Prints the options.
2. Computes the frequencies using Plinl (computeFrequency()).
3. Finds markers with MAF of 0, and saves them in a file (findSnpWithMaf0()).

Step11.flag_maf_zero.**parseArgs**(*argString=None*)
Parses the command line options and arguments.

Parameters **argString** (*list of strings*) – the options.

Returns A argparse.Namespace object created by the argparse module. It contains the values of the different options.

Options	Type	Description
--bfile	string	The input file prefix (Plink binary file).
--out	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see checkArgs()).

5.13 Hardy Weinberg Equilibrium Module

The usage of the standalone module is shown below:

```
$ pyGenClean_flag_hw --help
usage: pyGenClean_flag_hw [-h] --bfile FILE [--hwe FLOAT] [--out FILE]

Flag SNPs with Hardy-Weinberg disequilibrium.

optional arguments:
  -h, --help    show this help message and exit

Input File:
  --bfile FILE  The input file prefix (will find the plink binary files by
                appending the prefix to the .bim, .bed and .fam files,
                respectively.

Options:
  --hwe FLOAT   The Hardy-Weinberg equilibrium threshold. [default: 1e-4]

Output File:
  --out FILE    The prefix of the output files. [default: flag_hw]
```

5.13.1 Input Files

This module uses PLINK's binary file format (`bed`, `bim` and `fam` files) for the source data set (the data of interest).

5.13.2 Procedure

Here are the steps performed by the module:

1. Computes the number of markers in the input file.
2. Computes the Bonferroni threshold.
3. Runs Plink to find failed markers for HWE with the Bonferroni threshold.
4. Runs Plink to find failed markers for HWE with the default threshold.
5. Compares the two marker lists (Bonferroni and default threshold) and finds markers that are between the two thresholds.

5.13.3 Output Files

The output files of each of the steps described above are as follow (note that the output prefix shown is the one by default [*i.e.* `flag_hw`]):

1. No files are created for this step.
2. No files are created for this step.
3. One set of PLINK's binary file is created:
   - `flag_hw.threshold_X.Xe-X`: the data set containing only the markers that pass the HWE test (above the Bonferroni threshold).
4. One set of PLINK's binary file is created:

- `flag_hw.threshold_1e-4`: the data set containing only the markers that pass the HWE test (above the genome wide significance threshold of 1×10^{-4}). This value can be modified at the command line.

5. Three files are created:
   - `flag_hw.snp_flag_threshold_X.Xe-X`: the list of markers that failed HWE test for the Bonferroni threshold.
   - `flag_hw.snp_flag_threshold_1e-4`: the list of markers that failed HWE test for the genome wide significance threshold of 1×10^{-4}. This value can be modified at the command line.
   - `flag_hw.snp_flag_threshold_between_1e-4-X.Xe-X`: the list of markers that failed HWE test at a threshold between the Bonferroni and the genome wide significance thresholds, so that you can exclude only the ones that have a lower p value than the Bonferroni threshold.

5.13.4 The Algorithm

For more information about the actual algorithms and source codes (the `Step12.flag_hw` module), refer to the following sections.

Step12.flag_hw

exception `Step12.flag_hw.`**`ProgramError`**(*msg*)
An `Exception` raised in case of a problem.

Parameters **msg** (*string*) – the message to print to the user before exiting.

`Step12.flag_hw.`**`checkArgs`**(*args*)
Checks the arguments and options.

Parameters **args** (`argparse.Namespace`) – a `argparse.Namespace` object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the `ProgramError` class, a message is printed to the `sys.stderr` and the program exists with code 1.

`Step12.flag_hw.`**`compareBIMfiles`**(*beforeFileName*, *afterFileName*, *outputFileName*)
Compare two BIM files for differences.

Parameters

- **beforeFileName** (*string*) – the name of the file before modification.
- **afterFileName** (*string*) – the name of the file after modification.
- **outputFileName** (*string*) – the name of the output file (containing the differences between the `before` and the `after` files.

Returns the number of differences between the two files.

The `bim` files contain the list of markers in a given dataset. The `before` file should have more markers than the `after` file. The `after` file should be a subset of the markers in the `before` file.

`Step12.flag_hw.`**`computeHWE`**(*prefix*, *threshold*, *outPrefix*)
Compute the Hardy Weinberg test using Plink.

Parameters

- **prefix** (*string*) – the prefix of all the files.

- **threshold** (*string*) – the Hardy Weinberg threshold.
- **outPrefix** (*string*) – the prefix of the output file.

Uses Plink to exclude markers that failed the Hardy-Weinberg test at a specified significance threshold.

`Step12.flag_hw.`**`computeNumberOfMarkers`**(*inputFileName*)
Count the number of marker (line) in a BIM file.

Parameters **inputFileName** (*string*) – the name of the `bim` file.

Returns the number of marker in the `bim` file.

`Step12.flag_hw.`**`main`**(*argString=None*)
The main function.

Parameters **argString** (*list of strings*) – the options.

These are the steps performed by this module:

1. Prints the options of the module.
2. Computes the number of markers in the input file (`computeNumberOfMarkers()`).
3. If there are no markers, the module stops.
4. Computes the Bonferroni therhold ($0.05/\text{nbMarkers}$).
5. Runs Plink to find failed markers with the Bonferroni threshold.
6. Runs Plink to find failed markers with the default threshold.
7. Compares the `bim` files for the Bonferroni threshold.
8. Compares the `bim` files for the default threshold.
9. Computes the "in between" marker list, which is the markers from the default threshold and the Bonferroni threshold.

`Step12.flag_hw.`**`parseArgs`**(*argString=None*)
Parses the command line options and arguments.

Parameters **argString** (*list of strings*) – the options.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--bfile`	string	The input file prefix (binary Plink file).
`--hwe`	float	The Hardy-Weinberg equilibrium threshold.
`--out`	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see `checkArgs()`).

`Step12.flag_hw.`**`runCommand`**(*command*)
Run a command.

Parameters **command** (*list of strings*) – the command to run.

Tries to run a command. If it fails, raise a `ProgramError`. This function uses the `subprocess` module.

Warning: The variable `command` should be a list of stings (no other type).

5.14 Comparison with a Gold Standard Module

Explain the procedure here.

5.14.1 Output Files

Describe the output files here.

5.14.2 The Code

exception Misc.compare_gold_standard.**ProgramError**(*msg*)
An Exception raised in case of a problem.

Parameters **msg** (*string*) – the message to print to the user before exiting.

Misc.compare_gold_standard.**checkArgs**(*args*)
Checks the arguments and options.

Parameters **args** (argparse.Namespace) – a Namespace object containing the options of the program.

Returns True if everything was OK.

If there is a problem with an option, an exception is raised using the ProgramError class, a message is printed to the sys.stderr and the program exists with code 1.

Misc.compare_gold_standard.**check_fam_for_samples**(*required_samples*, *source*, *gold*)
Check fam files for required_samples.

Misc.compare_gold_standard.**computeFrequency**(*prefix*, *outPrefix*)
Compute the frequency using Plink.

Misc.compare_gold_standard.**compute_statistics**(*out_dir*, *gold_prefix*, *source_prefix*, *same_samples*, *use_sge*, *final_out_prefix*)
Compute the statistics.

Misc.compare_gold_standard.**exclude_SNPs_samples**(*inPrefix*, *outPrefix*, *exclusionSNP=None*, *keepSample=None*, *transpose=False*)
Exclude some SNPs and keep some samples using Plink.

Misc.compare_gold_standard.**extractSNPs**(*prefixes*, *snpToExtractFileNames*, *outPrefixes*, *runSGE*)
Extract a list of SNPs using Plink.

Misc.compare_gold_standard.**findFlippedSNPs**(*goldFrqFile1*, *sourceAlleles*, *outPrefix*)
Find flipped SNPs and flip them in the data1.

Misc.compare_gold_standard.**findOverlappingSNPsWithGoldStandard**(*prefix*, *gold_prefixe*, *out_prefix*, *use_marker_names=False*)
Find the overlapping SNPs in 4 different data sets.

Misc.compare_gold_standard.**flipSNPs**(*inPrefix*, *outPrefix*, *flipFileName*)
Flip SNPs using Plink.

Misc.compare_gold_standard.**illumina_to_snp**(*strand*, *snp*)
Return the TOP strand of the marker.

Function that takes a strand (TOP or BOT) and a SNP (e.g. : [A/C]) and returns a space separated AlleleA[space]AlleleB string.

Parameters

- **strand** (*str*) – Either "TOP" or "BOT"
- **snp** (*str*) – Either [A/C], [A/T], [G/C], [T/C], [A/G], [C/G], [T/A] or [T/G].

Returns The nucleotide for allele A and the nucleotide for allele B (space separated)

Return type str

`Misc.compare_gold_standard.`**`keepSamples`**(*prefixes*, *samplesToExtractFileNames*, *outPrefixes*, *runSGE*, *transpose=False*)

Extract a list of SNPs using Plink.

`Misc.compare_gold_standard.`**`parseArgs`**(*argString=None*)

Parses the command line options and arguments.

Returns A `numpy.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see `checkArgs()`).

`Misc.compare_gold_standard.`**`read_same_samples_file`**(*filename*, *out_prefix*)

Reads a file containing same samples.

`Misc.compare_gold_standard.`**`read_source_alleles`**(*file_name*)

Reads an allele file.

`Misc.compare_gold_standard.`**`read_source_manifest`**(*file_name*)

Reads Illumina manifest.

`Misc.compare_gold_standard.`**`renameSNPs`**(*inPrefix*, *updateFileName*, *outPrefix*)

Updates the name of the SNPs using Plink.

`Misc.compare_gold_standard.`**`runCommand`**(*command*)

Run a command.

5.15 Plink Utils

This module provides useful functions and scripts for efficient interactions with PLINK's output files. For example, the majority of PLINK's output files are spaced delimited, and are formated in such a way that it is "beautiful" to the human eye, but is a bit harder to parse using a script compared to tabulated files. The `PlinkUtils.createRowFromPlinkSpacedOutput()` function helps producing an array of all the fields for each line.

5.15.1 Comparing BIM files

Another example is the fact that when PLINK removes a certain amount of markers from the data file, it just gives the number of excluded markers, but not a list. The `PlinkUtils.compare_bim` module creates a list of markers that were removed from the original dataset when compared with the new one. Here is the usage of the standalone script:

```
$ pyGenClean_compare_bim --help
usage: pyGenClean_compare_bim [-h] --before FILE --after FILE [--out FILE]

Compare BIM file

optional arguments:
  -h, --help     show this help message and exit

Input File:
  --before FILE  The name of the bim FILE before modification.
  --after FILE   The name of the bim FILE after modification.

Output File:
  --out FILE     The prefix of the output files. [default: snp_removed]
```

5.15.2 Subsetting a dataset

A useful standalone script is the `PlinkUtils.subset_data` module. It helps in subsetting a dataset by keeping or removing a set of samples, and at the same time extracting or excluding a set of markers. The following standalone script is available for the user:

```
$ pyGenClean_subset_data --help
usage: pyGenClean_subset_data [-h] --ifile FILE [--is-bfile] [--is-tfile]
                              [--is-file] [--exclude FILE] [--extract FILE]
                              [--remove FILE] [--keep FILE] [--out FILE]

Subset genotype data using Plink.

optional arguments:
  -h, --help      show this help message and exit

Input File:
  --ifile FILE    The input file prefix. The format will be specified by --is-
                  bfile, --is-tfile or --is-file, for bfile, tfile and file,
                  respectively.
  --is-bfile      The file specified by --ifile is a bfile
  --is-tfile      The file specified by --ifile is a tfile
  --is-file       The file specified by --ifile is a file

Options:
  --exclude FILE  A file containing SNPs to exclude from the data set.
  --extract FILE  A file containing SNPs to extract from the data set.
  --remove FILE   A file containing samples (FID and IID) to remove from the
                  data set.
  --keep FILE     A file containing samples (FID and IID) to keep from the
                  data set.

Output File:
  --out FILE      The prefix of the output files. [default: subset]
```

The standalone script works with the three most used PLINK's format: pedfile, transposed and binary pedfiles. The `--is-bfile`, `--is-tfile` and `--is-file` options tell the standalone script what is the format of the input file. The output file format will be the same as the input one.

5.15.3 The Algorithm

For more information about the actual algorithms and source codes (the `PlinkUtils`, `PlinkUtils.compare_bim` and `PlinkUtils.subset_data` modules), refer to the following sections.

PlinkUtils

`PlinkUtils.`**`createRowFromPlinkSpacedOutput`**(*line*)
Remove leading spaces and change spaces to tabs.

Param line: a line from a Plink's report file.

Type line: string

Returns an array containing each field from the input line.

Plink's output files are usually created so that they are human readable. Hence, instead of separating fields using tabulation, it uses a certain amount of spaces to create columns. Using the `re` module, the fields are split.

```
>>> line = " CHR                SNP         BP    A1      A2"
>>> createRowFromPlinkSpacedOutput(line)
['CHR', 'SNP', 'BP', 'A1', 'A2']
```

`PlinkUtils.`**`get_version`**()
Returns the version of the module.

Returns (major, minor, micro)

PlinkUtils.compare_bim

exception `PlinkUtils.compare_bim.`**`ProgramError`**(*msg*)
An `Exception` raised in case of a problem.

Parameters **msg** (*string*) – the message to print to the user before exiting.

`PlinkUtils.compare_bim.`**`checkArgs`**(*args*)
Checks the arguments and options.

Parameters **args** (*argparse.Namespace*) – an object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the `ProgramError` class, a message is printed to the `sys.stderr` and the program exists with code 1.

`PlinkUtils.compare_bim.`**`compareSNPs`**(*before*, *after*, *outFileName*)
Compares two set of SNPs.

Parameters

- **before** (*set*) – the names of the markers in the `before` file.
- **after** (*set*) – the names of the markers in the `after` file.
- **outFileName** (*string*) – the name of the output file.

Finds the difference between two sets of markers, and write them in the `outFileName` file.

Note: A `ProgramError` is raised if:

1. There are more markers in the `after` set than in the `before` set.

2.Some markers that are in the `after` set are not in the `before` set.

`PlinkUtils.compare_bim.`**`main`**`()`
The main function of the module.

The purpose of this module is to find markers that were removed by Plink. When Plinks exclude some markers from binary files, there are no easy way to find the list of removed markers, except by comparing the two BIM files (before and after modification).

Here are the steps of this module:

1.Reads the BIM file before the modification (`readBIM()`).

2.Reads the BIM file after the modification (`readBIM()`).

3.Compares the list of markers before and after modification, and write the removed markers into a file (`compareSNPs()`).

Note: This module only finds marker that were removed (since adding markers to a BIM file usually includes a companion file to tell Plink which marker to add.

`PlinkUtils.compare_bim.`**`parseArgs`**`()`
Parses the command line options and arguments.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--before`	string	The name of the BIM file before modification.
`--after`	string	The name of the BIM file after modification.
`--out`	string	The prefix of the output files

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see `checkArgs()`).

`PlinkUtils.compare_bim.`**`readBIM`**`(`*`fileName`*`)`
Reads a BIM file.

Parameters **fileName** (*string*) – the name of the BIM file to read.

Returns the set of markers in the BIM file.

Reads a Plink BIM file and extract the name of the markers. There is one marker per line, and the name of the marker is in the second column. There is no header in the BIM file.

PlinkUtils.subset_data

exception `PlinkUtils.subset_data.`**`ProgramError`**`(`*`msg`*`)`
An `Exception` raised in case of a problem.

Parameters **msg** (*string*) – the message to print to the user before exiting.

`PlinkUtils.subset_data.`**`checkArgs`**`(`*`args`*`)`
Checks the arguments and options.

Parameters **args** (*argparse.Namespace*) – an object containing the options of the program.

Returns `True` if everything was OK.

If there is a problem with an option, an exception is raised using the `ProgramError` class, a message is printed to the `sys.stderr` and the program exists with code 1.

Note: Only one operation for markers and one operation for samples can be done at a time. Hence, one of `--exclude` or `--extract` can be done for markers, and one of `--remove` or `--keep` can be done for samples.

`PlinkUtils.subset_data.`**`main`**(*argString=None*)

The main function of the modile.

Parameters **argString** (*list of strings*) – the options.

Here are the steps:

1. Prints the options.
2. Subset the data (`subset_data()`).

Note: The type of the output files are determined by the type of the input files (*e.g.* if the input files are binary files, so will be the output ones).

`PlinkUtils.subset_data.`**`parseArgs`**(*argString=None*)

Parses the command line options and arguments.

Parameters **argString** (*list of string*) – the parameters.

Returns A `argparse.Namespace` object created by the `argparse` module. It contains the values of the different options.

Options	Type	Description
`--ifile`	string	The input file prefix.
`--is-bfile`	bool	The input file is a bfile
`--is-tfile`	bool	The input file is a tfile
`--is-file`	bool	The input file is a file
`--exclude`	string	A file containing SNPs to exclude from the data set.
`--extract`	string	A file containing SNPs to extract from the data set.
`--remove`	string	A file containing samples (FID and IID) to remove from the data set.
`--keep`	string	A file containing samples (FID and IID) to keep from the data set.
`--out`	string	The prefix of the output files.

Note: No option check is done here (except for the one automatically done by argparse). Those need to be done elsewhere (see `checkArgs()`).

`PlinkUtils.subset_data.`**`runCommand`**(*command*)

Runs a command.

Parameters **command** (*list of strings*) – the command to run.

If there is a problem, a `ProgramError` is raised.

`PlinkUtils.subset_data.`**`subset_data`**(*options*)

Subset the data.

Parameters **options** (*argparse.Namespace*) – the options.

Subset the data using either `--exclude` or `--extract``for markers or ``--remove` or `keep` for samples.

CHAPTER

SIX

APPENDIX

6.1 Result Summary Table

This table summurized information available from output files produced by *pyGenClean* during the data clean up procedure. Numbers correspond to number of lines in output files see *Proposed Protocol* for details. Only removed SNPs and IDs are indicated in the column SNPs and IDs, flagged SNPs or IDs are present in the n column.

Table 6.1: Summary information of the data clean up procedure.

Description	n	SNPs	IDs
Total number of SNPs in file received	2,379,855		
Total number of samples	494		
Number of duplicate samples	0		
Number of individuals with no genotype (failed)	0		
Number of SNPs with no physical position (chromosome and physical position = 0)	7,239	-7,239	
Number of INDEL	43	-43	
Number of replicate controls	5		
Number of replicate samples	0		
Number of duplicate SNPs (by chromosome and physical position)	5,643		
Duplicated SNPs by chromosome and physical position with the same allele (merge)	5,417	-5,147	
Number of duplicated SNP with <98% concordance	22	-22	
Completely failed SNPs	1	-1	
All heterozogous SNPs	0		
Number of individuals removed because they have more than 10% missing genotypes	5		-5
Number of SNPs removed because they have more than 2% missing value	128,562	-128,562	
Number of individuals removed because they have more than 2% missing genotypes	7		-7
Number of individuals with gender problem	1		
Number of SNPs with plate bias test P value below threshold of 1×10^{-7}	19		
Number of SNPs used for IBS analysis	73,651		
Number of duplicates pairs or twin	1		
Number of related pairs (including twins)	2		
Number of SNPs used for MDS analysis	80,262		
Number of individuals with ethnicity other than Caucasian as detected by MDS analysis	20		

Continued on next page

Table 6.1 – continued from previous page

Description	n	SNPs	IDs
Number of gender problems	1		-1
Number of related pairs	2		-2
Number of caucasian outliers	20		-20
Number of controls	5		-5
Number of heterozygote haploid genotypes set to missing (after correction of gender problems)	277,206		
Number of SNPs with MAF=0	602,480	-602,480	
Number of SNPS with HWE test P Value below threshold of 1×10^{-4} and higher than Bonferroni threshold	603		
Number of SNPS with HWE test below Bonferroni threshold	162	-162	
Total number of SNPs	**1,635,931**		
Total number of samples	**454**		

PYTHON MODULE INDEX

m

p

r

s

INDEX

E

F

G

I

K

M

P

R

S