

Université de Montréal

Cartographie génétique fine par le graphe de recombinaison ancestral

par

Fabrice Larribe

Département de mathématiques et de statistique

Faculté des arts et des sciences

Thèse présentée à la Faculté des études supérieures

en vue de l'obtention du grade de

Philosophiæ Doctor (Ph.D.)

en Statistique

août 2003

©Fabrice Larribe, 2003

Université de Montréal
Faculté des études supérieures

Cette thèse intitulée:

Cartographie génétique fine par le graphe de recombinaison ancestral

présentée par:

FABRICE LARRIBE

a été évaluée par un jury composé des personnes suivantes:

CHRISTIAN LÉGER

président-rapporteur

SABIN LESSARD

directeur de recherche

NICHOLAS SCHORK

codirecteur

JEAN-FRANÇOIS ANGERS

membre du jury

MONTGOMERY SLATKIN

examineur externe

NADIA EL-MABROUK

représentant du doyen de la FES

Résumé

Nous proposons une nouvelle méthode de cartographie génétique fine, basée sur le déséquilibre de liaison, qui utilise le graphe de recombinaison ancestral pour décrire l'histoire de séquences pouvant porter une mutation qui crée une maladie. Nous décrivons d'abord les méthodes existantes, et montrons que le processus de coalescence avec recombinaison peut être utilisé pour faire de la cartographie. La construction d'une équation de récurrence utilisée pour faire des inférences quant à la position de la mutation dans la séquence est décrite. Un algorithme de Monte Carlo combiné à un schéma d'échantillonnage pondéré local est développé afin d'obtenir la vraisemblance. Le comportement de la méthode est étudié, et nous l'évaluons à l'aide d'exemples simulés et réels. Nous montrons comment obtenir les temps entre les événements dans la reconstruction du graphe de recombinaison ancestral, ce qui nous permet de tenir compte des tailles de population variables. Nous étendons la méthode à différents types de marqueurs génétiques, et proposons diverses options afin de l'améliorer. Nous montrons que la méthode fonctionne bien sur des données simulées, bien que des variations non négligeables peuvent être observées dans les profils de vraisemblance. Les résultats sur des exemples réels où l'on connaît la position exacte de la mutation nous montrent que les estimations restent difficiles dans ces cas, bien que la méthode reste prometteuse.

Mots-clés

Arbre de recombinaison ancestral (ARG), cartographie génétique fine, processus de coalescence, déséquilibre de liaison, taille de population variable.

Summary

We present a multilocus gene mapping method based on linkage disequilibrium, which uses the ancestral recombination graph to model the history of sequences that may harbor an influential variant. We describe current fine mapping approaches, and show that the extension of the coalescent process to include recombination, the ancestral recombination graph, can be used for genetic fine mapping. We describe the construction of a recurrence equation used to make inferences about the location of a trait-influencing mutation and demonstrate how a Monte Carlo algorithm combined with a local importance sampling scheme can be used to obtain a likelihood curve. We explain how to simulate the timing of events in the coalescent in presence of recombination and mutation, which accommodates variable population size. We extend the method to different types of genetic markers, and propose some ideas to make it less computationally intensive. We study in detail the behavior of the method and, using simulated data, show that the method is working correctly despite some variability in the likelihood profiles. We provide real examples, which show that estimations are difficult in these cases, although the method seems to offer a great deal of promise.

Keywords

Ancestral recombination graph (ARG), fine genetic mapping, coalescent process, linkage disequilibrium, variable population size.

Résumé de vulgarisation

En génétique, l'un des aspects connaissant le plus d'intérêt à l'heure actuelle est de localiser les mutations de gènes jouant un rôle dans les maladies. C'est ce que l'on appelle "la cartographie génétique"; nous parlons de cartographie génétique fine si l'on étudie seulement une petite région du génome. Dans cette étude, nous avons développé une méthode de cartographie fine pour trouver la position d'une mutation causant une maladie dans un échantillon de cas et de témoins. L'échantillon est composé de séquences qui sont des suites ordonnées de marqueurs génétiques (c'est-à-dire, de variations de l'ADN à des positions données dans le génome). Nous utilisons le graphe de recombinaison ancestral, qui est une extension du processus de coalescence quand la recombinaison est considérée, pour modéliser l'histoire des séquences. Le graphe de recombinaison ancestral est un processus stochastique de naissance et de mort utilisé en mathématiques et statistique génétiques pour inférer des paramètres génétiques. La généalogie des séquences est modélisée en remontant dans le temps, en partant du temps d'échantillonnage jusqu'au temps de l'ancêtre commun de toutes les séquences. Trois types d'événements peuvent se produire dans l'histoire: un événement de coalescence quand deux séquences trouvent un ancêtre commun, un événement de mutation quand un marqueur d'une séquence mute vers un autre état, et un événement de recombinaison quand une séquence a deux ancêtres dans le passé. Comme le processus est stochastique, il est possible de décrire une équation de récurrence qui relie la probabilité de l'état de la chaîne à un autre dans un passé immédiat. Cette récurrence, qui peut aussi être vue comme de l'échantillonnage

pondéré, est alors utilisée pour faire des inférences sur la position de la mutation dans la séquence. À chaque étape du graphe, la probabilité de chaque événement est calculée, et un événement est choisi au hasard. Nous montrons comment cet algorithme de Monte Carlo peut être utilisé pour faire de la cartographie génétique. Aussi, nous montrons comment obtenir les temps entre les événements en présence de recombinaison, ce qui permet de modéliser des tailles de population variables. Nous étudions de quelle façon se comporte la méthode et l'évaluons par la suite et ce, à l'aide d'exemples simulés et réels. L'obtention des courbes de vraisemblance demande cependant beaucoup de temps de calcul: des jours voire des semaines sont nécessaires pour obtenir un estimé fiable de la vraisemblance. La méthode est par ailleurs étendue à différents types de marqueurs génétiques (SNPs et microsatellites) et diverses options sont proposées puis testées afin d'améliorer la méthode. Nous montrons que la méthode fonctionne bien sur des données simulées, bien que des variations non négligeables peuvent être observées dans les profils de vraisemblance. Les résultats sur des exemples réels où l'on connaît déjà la position exacte de la mutation nous montrent que les estimations restent difficiles dans des cas réels, bien que la méthode soit prometteuse. Une partie des résultats de cette recherche a été publiée dans *Theoretical Population Biology* en 2002 (Larribe *et al.*, 2002).

Table des matières

Résumé	iii
Summary	iv
Résumé de vulgarisation	v
Listes des tableaux	x
Listes des figures	xi
Avant-propos	1
1 Gènes et déséquilibre de liaison	5
1.1 Génétique	5
1.2 Études par linkage et par déséquilibre de liaison	11
1.3 Maladies complexes	14
2 De la cartographie génétique au processus de coalescence	16
2.1 Méthodes statistiques de cartographie génétique fine	16
2.1.1 Combinaison de statistiques à différents loci	18
2.1.2 Entre parcimonie et vraisemblance	20
2.1.3 Décroissement du partage d'haplotypes (DHS)	26
2.1.4 Reconstruction d'haplotypes ancestraux (AHR)	30
2.1.5 Approche de Rannala et Slatkin	33

2.2	Le processus de coalescence	41
2.3	Le graphe de recombinaison ancestral	47
2.4	Fondements théoriques de l'ARG	52
2.5	Méthodes d'estimation	59
2.5.1	Méthode de Griffiths et Tavaré	59
2.5.2	Amélioration de Fearnhead et Donnelly	62
2.5.3	Autres méthodes	65
3	Cartographie fine par le graphe de recombinaison ancestral	67
3.1	Introduction	67
3.2	Le modèle	70
3.3	Définition des probabilités de récurrence	77
3.4	Identités algébriques et échantillonnage pondéré	80
3.5	Un exemple illustratif	86
3.6	Intervalle de confiance	87
3.7	Description des ensembles de données	89
3.8	Détails de la méthode	93
3.9	Test de la méthode	97
3.10	Valeur conductrice	107
3.11	Méthode approximative pour les recombinaisons	111
3.12	Taille de population variable	118
4	Implantation informatique	124
4.1	Choix du langage de programmation	124
4.2	Algorithme	125
4.3	Exemple de reconstruction d'un ARG	132
4.4	Amélioration de l'algorithme précédent	136
4.5	Amélioration supplémentaire de l'algorithme	140
4.6	Programmation	143

4.7 Aspects numériques	143
4.8 Interface du programme	145
5 Améliorations et extensions de la méthode	146
5.1 Adaptation aux marqueurs microsatellites	146
5.2 Analyse des données de l'épilepsie myoclonique progressive (EPM1) .	156
5.3 Analyse des données de la fibrose kystique	158
5.4 Nombre maximum d'itérations dans un graphe	163
5.5 Ajustement des poids des recombinaisons	168
5.6 Recombinaisons et sites ancestraux	172
5.7 Vers une meilleure distribution proposée	176
5.8 Rééchantillonnage	186
5.8.1 Idée originale de Liu	186
5.8.2 Adaptation au graphe de recombinaison ancestral	189
5.8.3 Exemple	195
5.8.4 Effet du nombre de graphes utilisés pour le rééchantillonnage .	196
5.8.5 Tests de rééchantillonnage	198
Conclusion	202
Bibliographie	206
Annexes	212
A. Données de l'exemple E	213
B. Manuel d'instruction du programme	218
C. Programme informatique	236

Listes des tableaux

5.3.1. Estimés de r_T pour les données de la fibrose kystique par différentes méthodes de cartographie fine (IC: intervalle de confiance, %IC: proportion de l'IC par rapport à la séquence complète. Adapté de Morris <i>et al.</i> , 2002)	160
5.4.0. Évaluation du pourcentage de graphes rejetés ($\bar{\nu}$) et du gain de temps de calcul en modifiant le nombre d'itérations permis (ν) pour reconstruire un graphe pour les exemples B, C et D	165
5.7.2. Valeurs des couples n_j/n_k et n'_j/n'_k pour chacune des séquences de la figure 5.7.2, pour les quatre premières étapes de la reconstruction du graphe (- signifie 0).....	180

Listes des figures

1.1.1. Caryotype humain. Les différentes bandes de couleurs dépendent des différentes techniques de coloration utilisées pour visualiser les chromosomes durant la division cellulaire (mitose), spécifiquement durant la métaphase (adapté de “The Cytogenetics Gallery”, Département de pathologie, Université de Washington)	6
1.1.2. Illustration de la transmission des gènes dans une cellule familiale. Les carrés représentent les hommes, les cercles les femmes. Sous chacun des 9 individus est représentée une paire de chromosomes	7
1.1.3. Illustration de l’évolution d’un marqueur: (a) microsatellite, (b) SNP	8
1.1.4. Nous observons trois marqueurs bialléliques A, B, et C ayant respectivement les allèles A_1/A_2 , B_1/B_2 et C_1/C_2 . Un enjambement se produit entre les marqueurs A et B, résultant en une recombinaison. Deux enjambements se produisent entre les marqueurs B et C, mais si nous observons seulement les trois marqueurs A, B et C nous ne pouvons pas constater les enjambements car les deux haplotypes obtenus sont ‘ $A_1B_2C_2$ ’ et ‘ $A_2B_1C_1$ ’	10
1.2.1. Généalogie illustrant la transmission d’un gène au locus A, ayant pour allèles A_1 , A_2 , A_3 et A_4 , dont l’allèle A_2 est responsable d’une maladie. Les individus atteints sont en gris; les hommes sont représentés par des carrés, les femmes par des cercles. Les allèles B_1 , B_2 , B_3 et B_4 représentent les allèles au locus B, lié au premier. Le matériel génétique paternel est à gauche, alors que le maternel est à droite de chaque génotype	12
1.2.2. Illustration de l’évolution d’un segment chromosomique au cours du temps. À $t = t_0$, une mutation se produit sur l’un des chromosomes, et on peut suivre le partage chromosomique au cours du temps. Plus on avance dans le temps ($t_0 < t_1 < t_2$), plus les recombinaisons “cassent” le segment original et mélangent les séquences	13
2.1.1. Généalogie en étoile: les 12 séquences s^1 à s^{12} descendent directement d’un ancêtre commun (le MRCA, pour “most recent common ancestor”) 17	

2.1.2. Les blocs représentent des marqueurs, et les couleurs les allèles de ces marqueurs. La mutation (\bullet) est supposée être dans l'intervalle 4, entre les marqueurs 4 et 5. A est la racine. Par exemple, $A \rightarrow D$ et $L_1 \rightarrow E$ sont des recombinaisons, et les connections $A \rightarrow B$, $A \rightarrow C$, $A \rightarrow L_1$ se font par des événements de mutation ou de recombinaison. Comme il est impossible de connecter A et E par un simple événement, un haplotype latent L_1 est ajouté à l'arbre (Adaptée de Lam <i>et al.</i> , 2000)	21
2.1.3. Illustration d'une séquence, avec L marqueurs et $L - 1$ intervalles	21
2.1.4. Illustration du modèle: l'haplotype h_{obs} est ancestral autour de la mutation (\blacksquare , locus 0), du locus $-k$ jusqu'au locus j . Un locus ancestral est représenté par \blacksquare et un locus non ancestral par \square	26
2.1.5. Illustration de la mutation causant la maladie D , avec les trois marqueurs A , B , et C	31
2.1.6. (a) Une mutation M (\bullet) s'est produite au temps t_1 . Les descendances mutantes sont dénotées par les lignes hachurées. Les flèches indiquent les chromosomes échantillonnés. Le premier ancêtre commun des chromosomes porteurs de M apparaît au temps t_2 et est indiqué par un cercle vide. (b) Généalogie des quatre chromosomes porteurs de M échantillonnés de (a). Même si t_1 reste fixe, t_2 est plus récent que pour la population entière (Adapté de Rannala et Slatkin, 1998)	35
2.1.7. Schéma du nombre de chromosomes MA_1 et MA_2 lors d'un événement de coalescence	38
2.1.8. Schéma des données de DTD	40
2.1.9. Vraisemblance de la position de la mutation (Rannala et Slatkin, 1998). 41	
2.2.1. Réalisation d'un processus de coalescence pour $2N = 6$ sur dix générations. À gauche: toutes les relations entre les séquences sont indiquées. À droite: mise en évidence de la généalogie des six séquences de la dernière génération	42
2.2.2. Réalisation d'un processus de coalescence pour $2N = 6$ (suite de la fig. 2.2.1). On ignore la partie de la généalogie ne concernant pas les séquences de la dernière génération, et on peut représenter la généalogie des six séquences par une structure en arborescence (à gauche), menant à la représentation habituelle d'un arbre de coalescence (à droite)	43
2.2.3. Arbre de coalescence, avec l'indication des temps T_i ($i = 1, \dots, 6$), et des temps d'attente entre événements de coalescence W_i ($i = 2, \dots, 6$)	45
2.3.1. Illustration de la transmission du matériel génétique d'une séquence (fragment du chromosome maternel de l'individu 3, en noir) lors d'un événement de recombinaison dans le contexte de l'ARG	48
2.3.2. Généalogie familiale sur neuf générations, où le matériel génétique de quatre séquences de la dernière génération est suivi jusqu'à leur ancêtre commun. Deux recombinaisons et deux mutations (\bullet et \circ) se produisent dans ce matériel	49
2.3.3. Graphe de recombinaison ancestral résultant de la généalogie de la figure 2.3.2 pour l'échantillon de quatre séquences. Les séquences sont	

représentées par l'intervalle $[0, 1]$. Deux recombinaisons se produisent aux points 0.3 et 0.6. Les mutations sont \bullet et \circ	50
2.4.1. (a) Exemple de graphe de recombinaison ancestral. Les mutations sont représentées par des \bullet , avec l'indication de leur coordonnée. Une recombinaison se produit quand une arête se sépare en deux, alors qu'une coalescence se produit quand deux arêtes se confondent dans une. La coordonnée du point de recombinaison est également indiquée. (b)–(f) arbres de coalescence marginaux du graphe (a). Les arêtes représentées en pointillé gris n'appartiennent pas aux arbres marginaux	53
3.1.1. Un exemple d'ARG. Symboles: \blacksquare marqueur ancestral primitif; \blacksquare marqueur ancestral mutant; \square marqueur non ancestral	69
3.2.1. Configuration d'une séquence si le TIM (\blacksquare) est à la position m . On définit par r_l la distance entre le TIM et le marqueur immédiatement à gauche, et r_r la distance entre le TIM et le marqueur immédiatement à droite	72
3.2.2. Arbres de coalescence partiels pour chacun des quatre marqueurs du graphe de recombinaison ancestral de la figure 3.1.1	73
3.2.3. Exemple de deux séquences \mathbf{s}^1 et \mathbf{s}^2 de cinq marqueurs numérotés $1, \dots, 5$. (\square représente un marqueur ancestral, \square un marqueur non ancestral)	75
3.2.4. Exemples de coalescences possibles entre séquences de différents types (\blacksquare représente un site ancestral primitif, \blacksquare représente un site ancestral mutant, et \square un site non ancestral)	76
3.5.1. Exemple de résultat produit par la méthode	86
3.6.1. Exemple de calcul d'intervalle de confiance approximatif pour les données de l'exemple introduit dans la section précédente: (a) logarithme de la vraisemblance, (b) vraisemblance relative	88
3.7.1. Exemple A de données simulées: multiplicité de chaque séquence i (n_i) pour $i = 1, \dots, 7$, les séquences 3, 4 et 6 étant pour des cas, les autres pour les témoins. Le TIM occupe en fait la troisième position	90
3.7.2. Exemple B de données simulées: multiplicité de chaque séquence i (n_i) pour $i = 1, \dots, 6$, les séquences 1, 3, 4 et 5 étant pour des cas, les autres pour les témoins. Le TIM occupe en fait la troisième position	91
3.7.3. Exemple C de données simulées: multiplicité de chaque séquence i (n_i) pour $i = 1, \dots, 5$, les séquences 1, 4 et 5 étant pour des cas, les autres pour les témoins. Le TIM occupe en fait la seconde position	91
3.7.4. Exemple D de l'épilepsie myoclonique progressive. Multiplicité de chaque séquence i (n_i) pour $i = 1, \dots, 10$, toutes les séquences sont des cas. Le TIM occupe en fait la troisième position	92
3.8.1. Résultats d'un premier ensemble de simulations pour les données de l'exemple B: (a) itérations de 1 à 100M, (b) itérations de 1M à 100M . . .	94
3.8.2. Résultats d'un second ensemble de simulations pour les données de l'exemple B: (a) itérations de 1 à 100M, (b) itérations de 1M à 100M . . .	95
3.8.3. Résultats d'un premier ensemble de simulations pour les données de l'exemple C: (a) itérations de 1 à 50M, (b) itérations de 1M à 50M	96

3.8.4. Résultats d'un second ensemble de simulations pour les données de l'exemple C: (a) itérations de 1 à 100M, (b) itérations de 1M à 100M ...	97
3.9.1. Courbes de vraisemblance pour l'exemple A: (a) 1M d'itérations chaque, (b) 5M d'itérations chaque, (c) 10M d'itérations chaque et (d) 20M d'itérations chaque. Les courbes en gras sont pour les itérations cumulées	98
3.9.2. Courbes de vraisemblance pour l'exemple B: (a) 1M d'itérations chaque, (b) 5M d'itérations chaque et (c) 20M d'itérations chaque. Les courbes en gras sont les courbes cumulées	99
3.9.3. Courbes de vraisemblance pour le second intervalle de l'exemple B: (a) 1M d'itérations chaque, (b) 5M d'itérations chaque, c) 20M d'itérations chaque et (d) 100M d'itérations chaque	100
3.9.4. Courbes de vraisemblance pour l'exemple C: (a) 1M d'itérations chaque, (b) 5M d'itérations chaque, (c) 10M d'itérations chaque et (d) 100M d'itérations chaque. Les courbes en gras sont pour les itérations cumulées	102
3.9.5. Courbes de vraisemblance pour l'exemple C: (a) 50m d'itérations chaque, (b) 100m d'itérations chaque, (c) 500m d'itérations chaque, (d) 1M d'itérations chaque, (e) 2M d'itérations chaque et (f) 5M d'itérations chaque	104
3.9.6. Courbes de vraisemblance pour l'exemple B: (a) 50m d'itérations chaque, (b) 100m d'itérations chaque, (c) 500m d'itérations chaque, (d) 1M d'itérations chaque, (e) 5M d'itérations chaque et (f) 25M d'itérations chaque	105
3.9.7. Courbes de vraisemblance pour l'exemple D: (a) 100m d'itérations chaque, (b) 250m d'itérations chaque, (c) 1M d'itérations chaque, (d) 50M d'itérations chaque; (a), (b) et (c) $u = 6 \times 10^{-5}$, (d) $u = 8 \times 10^{-3}$	106
3.10.1. Deux estimations de la vraisemblance avec 1M d'itérations pour 120 valeurs conductrices pour l'exemple C.....	108
3.10.2. Deux estimations de la vraisemblance avec 2M d'itérations pour 120 valeurs conductrices pour l'exemple C.....	108
3.10.3. Deux estimations de la vraisemblance avec 5M d'itérations pour 120 valeurs conductrices pour l'exemple C.....	109
3.10.4. Deux estimations de la vraisemblance avec 1M d'itérations pour 100 valeurs conductrices pour l'exemple B.....	110
3.10.5. Deux estimations de la vraisemblance avec 5M d'itérations pour 100 valeurs conductrices pour l'exemple B.....	110
3.10.6. Deux estimations de la vraisemblance avec 10M d'itérations pour 100 valeurs conductrices pour l'exemple B.....	111
3.11.1. Courbes de vraisemblance pour l'exemple A: (a), (b) et (c) méthode originale; (d), (e) et (f) méthode approximative; (a) et (d) 1M d'itérations, (b) et (e) 5M d'itérations, (c) et (f) 10M d'itérations.....	115
3.11.2. Courbes de vraisemblance pour l'exemple B: (a), (b) et (c) méthode originale; (d), (e) et (f) méthode approximative; (a) et (d) 1M d'itérations, (b) et (e) 5M d'itérations, (c) et (f) 20M d'itérations.....	116

3.11.3. Courbes de vraisemblance pour l'exemple C: (a), (b) et (c) méthode originale; (d), (e) et (f) méthode approximative; (a) et (d) 1M d'itérations, (b) et (e) 10M d'itérations, (c) et (f) 20M d'itérations.....	117
3.12.1. Densités pour les trois cas (a), (b) et (c). La ligne pleine est pour s_c et la ligne hachurée pour s_{crm}	123
4.2.1. Schéma d'insertion du TIM (■) dans la séquence des marqueurs observés de position connue (□). Les chiffres indiquent des distances entre loci. (a) Données brutes: 3 marqueurs. Modification des séquences et des distances pour l'évaluation de la vraisemblance dans l'intervalle entre marqueurs (b) un (c) deux.....	127
4.2.2. Illustration de l'effet du nombre de points pour lesquels la vraisemblance est calculée: (a) NbCan est 50, (b) NbCan est 200	131
4.3.1. Le graphe de recombinaison ancestral correspondant aux calculs effectués dans la section 4.3	136
4.5.1. Illustration des matrices pour $L = 3$. Sont présentés de gauche à droite le numéro unique d'identification de chacune des 25 séquences possibles (Id), les 25 séquences elles-mêmes (THap), la matrice d'information sur les coalescences (TCoa), la matrice d'information sur les mutations (TMu), et la matrice d'information du matériel ancestral pour les recombinaisons (TAnc: γ et κ). Sont omis de cette figure pour des raisons d'espace, mais calculés cependant, les vecteurs contenant l'information sur le contenu ancestral de chaque séquence, a et b (Ta et Tb), et les matrices contenant l'information sur les parents des séquences recombinantes à gauche (TReL) et à droite (TReR). Le symbole “-” indique un marqueur non ancestral pour THap, et un événement impossible pour TCoa et TMu.....	141
5.1.1. Illustration d'une généalogie de séquences suivie à l'aide de marqueurs microsatellites. Les nombres de répétitions sont représentés par des couleurs.....	147
5.1.2. Illustration d'une généalogie de séquences suivie à l'aide de marqueurs binaires à faible taux de mutation, dans le cadre d'un modèle où les mutations sont non récurrentes	148
5.1.3. Illustration d'une généalogie de séquences suivie à l'aide de marqueurs microsatellites: (a) généalogie réelle, (b) généalogie sans recombinaison.....	149
5.2.1. Exemple D de l'épilepsie myoclonique progressive: pour chaque séquence i (0 à 14), multiplicité $[n_i]$, et nombre de répétitions aux cinq marqueurs	156
5.2.2. Courbes de vraisemblance pour l'exemple D: (a) McPeck et Strahs (1999), et (b) Xiong et Guo (1997).....	157
5.2.3. Courbes de vraisemblance pour l'exemple D: (a) méthode originale avec 500m d'itérations chaque et $u_m = 1 \times 10^{-3}$, (b) méthode approximative avec 1M d'itérations chaque et $u_m = 1 \times 10^{-3}$, (c) méthode originale avec 500m d'itérations chaque et $u_m = 1 \times 10^{-4}$, (d) méthode approximative avec 1M d'itérations chaque et $u_m = 1 \times 10^{-4}$	158

5.3.1. Répartition des 23 marqueurs, représentés par des barres verticales grises, sur une échelle en Mb.....	159
5.3.2. Courbes de vraisemblance pour l'exemple E: (a) McPeck et Strahs (1999), et (b) Xiong et Guo (1997) (porteurs de $\Delta F508$ seulement)....	161
5.3.3. Deux courbes de vraisemblance pour l'exemple E, basées sur 100m d'itérations, en incluant seulement les séquences porteuses de $\Delta F508$..	162
5.3.4. Courbes de vraisemblance pour l'exemple E sur une partie seulement de la séquence, basées sur (a) 100m d'itérations chaque, (b) 250m d'itérations chaque.....	163
5.4.1. Courbes de vraisemblance pour l'exemple B, pour les 10 niveaux de ν du tableau 5.4.2: a) 1M d'itérations, b) 5M d'itérations.....	166
5.4.2. Courbes de vraisemblance pour l'exemple C, pour les 10 niveaux de ν du tableau 5.4.2: a) 1M d'itérations, b) 5M d'itérations.....	167
5.4.3. Courbes de vraisemblance pour l'exemple D, pour les 7 niveaux de ν du tableau 5.4.2: a) 100m d'itérations, b) 1M d'itérations.....	168
5.5.1. Courbes de vraisemblance pour l'exemple A avec 1M d'itérations et leur courbe cumulée, pour (a) $\xi = 1$, (b) $\xi = 0.85$, (c) $\xi = 0.70$ et (d) $\xi = 0.55$	169
5.5.2. Courbes de vraisemblance pour l'exemple B avec 5M d'itérations et leur courbe cumulée, pour (a) $\xi = 1$, (b) $\xi = 0.85$, (c), $\xi = 0.70$ et (d) $\xi = 0.55$	170
5.5.3. Courbes de vraisemblance pour l'exemple C avec 5M d'itérations et leur courbe cumulée, pour (a) $\xi = 1$, (b) $\xi = 0.85$, (c) $\xi = 0.70$ et (d) $\xi = 0.55$	171
5.6.1. Exemple de graphe de recombinaison ancestral: à gauche avec la méthode usuelle, et à droite en supprimant les loci qui perdent leur état polymorphe.....	173
5.6.2. Exemple fictif illustrant la procédure qui interdit certaines recombinaisons. Les événements possibles à l'étape 2 sont à droite. M est le vecteur informatif des mutations, et R indique les intervalles où les recombinaisons sont permises.....	174
5.6.3. Courbes de vraisemblance pour l'exemple B: (a) et (c) méthode originale, (b) et (d) procédure limitant les recombinaisons. (a) et (b) 1M d'itérations pour chaque courbe et la courbe cumulée, (c) et (d) 5M d'itérations pour chaque courbe et la courbe cumulée.....	175
5.6.4. Courbes de vraisemblance pour l'exemple C: (a) et (c) méthode originale, (b) et (d) procédure limitant les recombinaisons. (a) et (b) 1M d'itérations pour chaque courbe et la courbe cumulée, (c) et (d) 5M d'itérations pour chaque courbe et la courbe cumulée.....	176
5.7.1. Une recombinaison dans le second intervalle de la séquence s^2 conduit à cette généalogie (voir texte).....	178
5.7.2. Illustration d'un graphe de recombinaison ancestral montrant un événement de recombinaison dans le second intervalle.....	179
5.7.3. Illustration d'un arbre pour stocker l'information des haplotypes.....	181

5.7.4. Courbes de vraisemblance pour l'exemple A: (a), (b) et (c) procédure originale; (d), (e) et (f) procédure simplifiée; (a) et (d) 16m et 10m d'itérations respectivement; (b) et (e) 160m et 100m d'itérations respectivement; (c) et (f) 1.6M et 1M d'itérations respectivement	182
5.7.5. Courbes de vraisemblance pour l'exemple B: (a), (b) et (c) procédure originale; (d), (e) et (f) procédure simplifiée; (a) et (d) 16m et 10m d'itérations respectivement; (b) et (e) 160m et 100m d'itérations respectivement; (c) et (f) 1.6M et 1M d'itérations respectivement	183
5.7.6. Courbes de vraisemblance pour l'exemple C: (a), (b) et (c) procédure originale; (d), (e) et (f) procédure simplifiée; (a) et (d) 17.5m et 10m d'itérations respectivement; (b) et (e) 175m et 100m d'itérations respectivement; (c) et (f) 1.75M et 1M d'itérations respectivement	184
5.7.7. Courbes de vraisemblance pour l'exemple C: (a) procédure originale, 8.75M d'itérations; (b) procédure simplifiée, 5M d'itérations	185
5.8.1. Tiré de Liu (2001). Exemple d'estimation de θ pour un processus de coalescence pur; (a) sept estimations indépendantes par la méthode de Griffiths et Tavaré (1994b); (b) cinq estimations indépendantes avec rééchantillonnage par la méthode de Liu (2001)	188
5.8.2. Corrélations r_n^1 entre \bar{w}_n et \bar{w}_1 , pour $n = 2, \dots, 17$, sur 1000 graphes de l'exemple D	190
5.8.3. Évolution de la distribution et de la corrélation des poids pendant la reconstruction du graphe, à partir des données de l'exemple D	191
5.8.4. Évolution de la distribution et de la corrélation des poids pendant la construction du graphe à partir des données de l'exemple B	192
5.8.5. Évolution de la distribution et de la corrélation des poids pendant la construction du graphe à partir des données de l'exemple C	192
5.8.6. Deux échantillonnages de 50m d'itérations, avec $m' = 50m$, $n_S = 5$, et deux rééchantillonnages de 50m d'itérations pour chacun des échantillonnages. La courbe en gras est la courbe cumulée	196
5.8.7. À partir d'un échantillon de 50m de graphes de l'exemple B, rééchantillonnage basé sur des valeurs de m' de 50m, 40m, 30m, 20m, 10m, 5m et 1m: (a) $n_S = 3$, (b) $n_S = 4$, (c) $n_S = 5$	197
5.8.8. À partir d'un échantillon de 50m de graphes de l'exemple C, rééchantillonnage basé sur des valeurs de m' de 50m, 40m, 30m, 20m, 10m, 5m et 1m: (a) $n_S = 2$, (b) $n_S = 4$, (c) $n_S = 6$	197
5.8.9. Échantillonnage et rééchantillonnage de 1M d'itérations pour l'exemple B: (a) $n_S = 2$, (b) $n_S = 3$, (c) $n_S = 4$ ($m' = 1m$)	198
5.8.10. Échantillonnage et rééchantillonnage de 500m d'itérations pour l'exemple C: (a) $n_S = 2$, (b) $n_S = 3$, (c) $n_S = 4$ ($m' = 5m$)	198
5.8.11. Rééchantillonnage pour l'exemple E ($m' = 1m$): (a) et (b) $n_S = 18$; (c) et (d) $n_S = 9$; (a) et (c) $m = 10m$; (b) et (d) $m = 25m$. Les courbes noires sont les courbes cumulées	199

- 5.8.12. Rééchantillonnage pour l'exemple D ($m' = 1m$): (a) et (b) $n_S = 5$; (c) et (d) $n_S = 10$; (a) et (c) $m = 100m$; (b) et (d) $m = 250m$. Les courbes noires sont les courbes cumulées 200

Listes des symboles

a	$a = \sum_{i=1}^d n_i A_i $. Nombre total de marqueurs où une mutation est possible ($n \leq a \leq nL$).
b	Longueur totale du matériel ancestral de l'échantillon ($0 \leq b \leq nr$).
β	$\beta = b/(nr)$.
c_i	Proportion de la longueur d'une séquence i pour laquelle une recombinaison affecte le matériel ancestral.
C_i	Événement de coalescence de deux séquences identiques de type i .
C_{ij}^k	Événement de coalescence de deux séquences i and j vers la séquence k .
cM	Le centiMorgan est une unité de distance génétique: $1cM \approx 1000kb = 1Mb$.
$\underline{\Delta}_m$	Valeur minimum possible pour le locus m dans le modèle microsatellite.
$\overline{\Delta}_m$	Valeur maximum possible pour le locus m dans le modèle microsatellite.
$f_Z(z)$	Densité du point de recombinaison.
\mathbf{H}_τ	Ensemble de séquences $\mathbf{s} = (s_1, \dots, s_L)$.
$\mathbf{H}_\tau + R_i^{jk}(p)$	\mathbf{H}_τ modifié par un événement de recombinaison dans l'intervalle p à l'étape $\tau + 1$.
γ_i	Intervalle où le matériel ancestral commence pour une séquence de type i .
κ_i	Intervalle où le matériel ancestral finit pour une séquence de type i .
h	Pas d'une mutation dans le modèle microsatellite.
i	Habituellement, utilisé pour un type de séquence.
K	Nombre d'itérations d'une simulation.
L	Nombre de marqueurs.
$L - 1$	Nombre de marqueurs de position connue.
m	Habituellement, un index sur les loci $(1, \dots, L)$.
p	Indice d'un intervalle.

ϕ	Pas maximum d'une mutation dans le modèle microsatellite.
r	Probabilité de recombinaison par séquence par génération.
r_p	Distance entre les loci x_p et x_{p+1} .
r_r	Distance entre le TIM et le marqueur immédiatement à droite.
r_l	Distance entre le TIM et le marqueur immédiatement à gauche.
r_T	Position du TIM (le premier marqueur est la référence).
$R_i^{jk}(p)$	Recombinaison d'une séquence i vers les séquences j and k dans l'intervalle p .
\mathbf{s}	Une séquence (s_1, \dots, s_L)
\mathbf{s}^i	La séquence i , (s_1^i, \dots, s_L^i)
\mathbf{s}_m^i	L'allèle m de la séquence i
TIM	La mutation causant la maladie (Trait-Influencing Mutation).
Θ	Ensemble des paramètres (r_T, θ) .
θ	Taux de mutation par séquence par génération.
Θ_0	Ensemble des paramètres utilisés pour construire un graphe: (r_{0T}, θ_0) .
u	Probabilité de mutation par séquence par génération.
v_m^h	Probabilité d'une mutation de taille h au locus m .
x_m	Coordonnée du $m^{\text{ième}}$ locus.

Remerciements

J'ai commencé l'école à l'âge de trois ans, et je suis resté sur les bancs pendant 27 courtes années. Tout ce temps pour apprendre, tout ce temps pour se rendre compte à la toute fin, que l'on ne sait pas grand chose. Mais ce n'était pas du temps perdu, loin de là. J'ai pris beaucoup de plaisir à étudier, et notamment à travailler sur cette thèse. J'ai fait ce que je voulais faire, et je suis extrêmement reconnaissant à tous ceux qui m'ont aidé, d'une façon ou d'une autre, à y parvenir.

Je remercie sincèrement mon directeur de thèse, Sabin Lessard, pour m'avoir beaucoup appris et m'avoir supervisé avec rigueur. Merci à mon co-directeur, Nik Schork, pour m'avoir fait découvrir la statistique génétique et m'avoir accompagné dans ce projet. Je remercie Denis Gauvreau et Michael Dennis pour m'avoir fait confiance. Merci aussi à mes superviseurs successifs de la compagnie, dont Isabel Fortier, Pierre Chrétien et Mario Filion. Merci à mes "collègues" John, Hélène, Nancy, Carole pour avoir répondu à mes mille et une questions en matière de génétique. Un énorme merci à Jean-François pour le support, l'aide et le temps-machine qu'il m'a généreusement offert. Ce travail a également bénéficié de discussions informelles avec Bob Griffiths, que je remercie beaucoup.

Il me faut évidemment remercier le Fonds pour la Formation de chercheurs et l'aide à la recherche (FCAR), le Conseil de recherches en sciences naturelles et en génie du Canada (CRSNG), Algène Biotechnologies, SignalGene, Sabin Lessard, Nicolas J. Schork et Serge Tardif pour le support financier et le matériel nécessaire à la réalisation de cette thèse.

Enfin, il y a les autres, ceux qui font du travail de soutien. Merci à vous tous.

Des remerciements spéciaux à mes lecteurs et correcteurs, Annick et Yvon. Un remerciement “de fond” à Carole pour son soutien précieux durant toutes mes études dans le nouveau monde. Un remerciement tout spécial pour Annick pour “tout”.

Avant-propos

Un des aspects les plus intéressants de la recherche génétique actuelle consiste à localiser les gènes qui causent les maladies. Ces gènes sont habituellement défectueux dû à une ou plusieurs mutations. Nous nommerons une telle mutation un TIM, pour *trait-influencing mutation*. D'abord, un échantillon de la population sous étude est retenu; cet échantillon est constitué de séquences considérées comme des séries de marqueurs génétiques (c'est-à-dire, des variations de l'ADN à des positions données dans le génome). Pour localiser ces gènes défectueux, deux approches sont possibles: le linkage et le déséquilibre de liaison. Le linkage utilise des modèles de transmission des chromosomes dans des généalogies familiales dans le but de tester différentes hypothèses sur la position du gène. Cependant, le faible nombre de générations des généalogies familiales ne permet qu'une faible précision des estimés, à cause du petit nombre de recombinaisons présentes dans des généalogies familiales. La recombinaison est le phénomène fondamental qui nous permet de faire de la cartographie génétique: de générations en générations, c'est par le processus de recombinaison que les chromosomes se mélangent, laissant ainsi des traces nous permettant de construire des modèles de probabilité de transmission des chromosomes. Sans recombinaison, il n'y aurait pas de variation, et donc pas de cartographie (notons cependant que de nouvelles méthodes, basées sur la biologie moléculaire, se développent et elles ne sont pas basées sur ces concepts). Pour pallier au manque de résolution des études de linkage, on peut utiliser le déséquilibre de liaison (*linkage disequilibrium*). Celui-ci réfère à l'association non aléatoire entre des allèles à différents loci liés. Pour l'illustrer,

supposons qu'une mutation se soit produite chez un ancêtre dans le passé voilà quelques dizaines de générations, et que nous échantillonnions aujourd'hui des individus atteints de la maladie causée par le TIM en question. Si l'on suppose que la maladie est rare (et simple), tous les cas vont partager non seulement le TIM en cause, mais également des segments d'ADN dans la région de la mutation. Plus la population est vieille, plus ce mélange sera grand. Évidemment, d'autres concepts, comme les mutations, entrent en cause dans la transmission du bagage génétique. C'est ce que nous décrirons dans le premier chapitre de la thèse.

Depuis quelques années, différents auteurs ont proposés des méthodes pour étudier le déséquilibre de liaison et faire de la cartographie fine, c'est-à-dire, chercher et identifier précisément la position du TIM à l'intérieur d'une région chromosomique, qui aura elle-même été identifiée au préalable par linkage, par exemple. Comme nous travaillons sur une population, la généalogie de la population n'est habituellement pas connue; il faut l'inférer. Ces méthodes vont donc construire des modèles décrivant la généalogie des séquences échantillonnées, puis développer des probabilités de transmission du bagage génétique sur ces généalogies en tenant compte habituellement des deux événements majeurs affectant la transmission de l'ADN: les mutations et les recombinaisons. Comme nous le verrons dans le second chapitre, la plupart des méthodes de cartographie fine supposent des généalogies qui décrivent de façon simpliste l'histoire des chromosomes, par des généalogies en étoile, ignorant ainsi la dépendance entre les séquences, ou mieux, par des arbres. Cependant, quand un événement de recombinaison se produit, une séquence provient de deux parents dans le passé, et l'histoire réelle de la séquence ressemble plus à un graphe qu'à un arbre. D'autre part, certaines de ces méthodes sont basées sur des combinaisons de statistiques à différents marqueurs et n'utilisent pas toute l'information disponible. Pour pallier à cette faiblesse, des méthodes multilocus utilisant la dépendance entre les marqueurs ont été développées.

Il existe par ailleurs un moyen simple de modéliser l'histoire des séquences:

il s'agit du processus de coalescence. Avec de simples hypothèses dont la plupart peuvent être allégées, le processus de coalescence est un processus stochastique qui nous permet de simuler un arbre qui décrit l'histoire des séquences observées en partant de l'échantillon et en remontant jusqu'à l'ancêtre commun de toutes ces séquences. Deux types d'événements sont alors considérés: des événements de coalescence, quand deux séquences trouvent un ancêtre commun, et des événements de mutation quand un marqueur d'une séquence mute vers un autre état. Le processus de coalescence est un outil utilisé pour inférer certains paramètres, comme des taux de mutation par exemple. De plus, la recombinaison peut être incorporée dans le processus de coalescence; un graphe est alors construit au lieu d'un arbre, et l'on obtient le *graphe de recombinaison ancestral* (ARG). Nous présenterons dans le chapitre 2 le processus de coalescence, les méthodes d'inférence que l'on peut utiliser, et le graphe de recombinaison ancestral. Nous nous proposons dans ce travail de développer une méthode de cartographie fine basée sur le graphe de recombinaison ancestral.

Nous décrirons donc le modèle utilisé dans le chapitre 3. Nous montrerons comment construire une équation de récurrence reliant l'état de la généalogie à une étape dans le passé vers une autre étape; nous montrerons comment utiliser cette équation de récurrence pour faire des inférences sur la position du TIM. Nous étudierons en détail le comportement de la méthode proposée, et nous l'évaluerons à l'aide d'exemples. Nous développerons ensuite des équations pour simuler les temps auxquels les événements se produisent; cela nous permettra de modéliser des tailles de population variables.

Puisque la méthode utilise des simulations de Monte Carlo, l'aspect informatique joue un rôle important, d'autant plus que, comme nous le verrons, la méthode peut être très exigeante en temps de calcul. Le chapitre 4 décrira donc l'implantation informatique de la méthode: l'algorithme développé, et les améliorations possibles.

Le chapitre 5 sera consacré à diverses extensions et améliorations de la méthode.

Nous développerons un modèle pour des marqueurs microsatellites, ce qui nous permettra de montrer des résultats sur des données réelles, puis nous proposerons diverses améliorations possibles pour diminuer le temps de calcul ou réduire la variabilité des courbes de vraisemblance estimées. Nous considérerons enfin le rééchantillonnage, afin de voir s'il peut améliorer les inférences.

Chapitre 1

Gènes et déséquilibre de liaison

Nous présentons dans ce chapitre une brève introduction aux bases de la génétique, dont la recombinaison, qui sont des notions nécessaires à la compréhension du travail. Nous définirons le déséquilibre de liaison, phénomène qui nous permet de faire de la cartographie génétique, et nous aborderons le concept de maladie complexe, enjeu de la cartographie génétique actuelle.

1.1. Génétique

Nous nous situons ici dans un contexte de génétique humaine et nous ignorerons donc les particularités génétiques des autres êtres vivants, bien que les travaux présentés ici peuvent s'appliquer avec quelques modifications à la plupart des êtres vivants.

L'être humain est constitué de 40 000 milliards de cellules. Dans le noyau de la plupart de ces cellules, nous retrouvons la totalité de notre patrimoine génétique.

Celui-ci est constitué de 46 chromosomes, en fait, 23 paires de chromosomes: 22 paires d'autosomes, et deux chromosomes sexuels: une paire XX pour une fille, et une paire XY pour un garçon (chez un garçon, le X étant transmis par la mère et le Y par le père). Un caryotype humain est représenté à la figure 1.1.1; il montre une représentation photographique des 22 autosomes et des deux chromosomes sexuels; on peut voir que les chromosomes ont des longueurs très différentes: le chromosome 1 est environ 4 fois plus grand que le plus petit des chromosomes, le 21. Chaque paire de chromosomes contient deux chromosomes dit homologues; l'un vient du père, et l'autre de la mère. Une version de chacun de nos gènes se trouve sur chacun des chromosomes d'une même paire, ce qui veut dire qu'un gène est présent en double exemplaire; notons que ces deux versions peuvent être identiques ou différentes, se compléter ou s'affronter. Les humains possédant

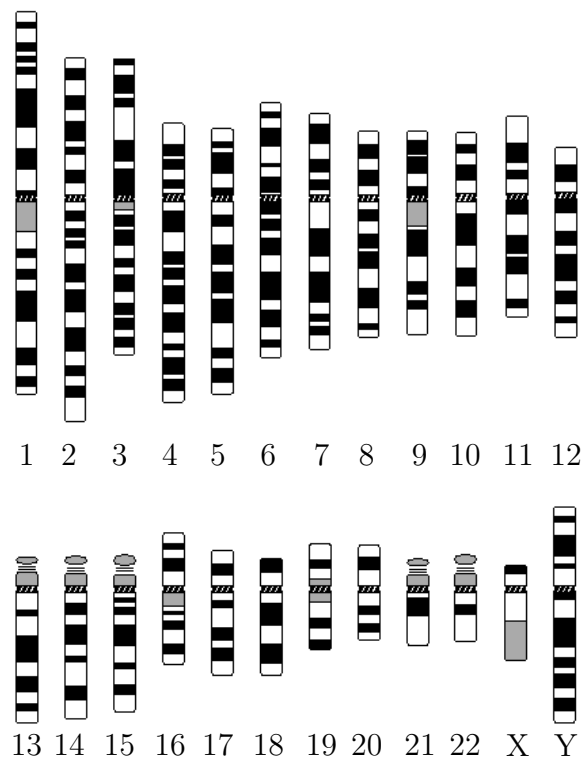


Figure 1.1.1. Caryotype humain. Les différentes bandes de couleurs dépendent des différentes techniques de coloration utilisées pour visualiser les chromosomes durant la division cellulaire (mitose), spécifiquement durant la métaphase (adapté de “The Cytogenetics Gallery”, Département de pathologie, Université de Washington).

deux versions de chaque gène, ils sont dits diploïdes, alors que les espèces ne possédant qu'une seule version sont quant à elles appelées haploïdes. Les chromosomes sont constitués d'acide désoxyribonucléique (ADN), ce dernier étant formé avec quatre types de nucléotide A, C, G, T (adénine, cytosine, guanine, thymine). Les gènes correspondent à des suites de quelques milliers à deux millions de nucléotides qui sont traduites via le code génétique en suites d'acides aminés qui forment des protéines ou des enzymes.

Notons enfin que nous aurions entre 30 000 et 100 000 gènes répartis sur 10% de l'ADN; la partie non codante de l'ADN (les 90% restants) demeure encore une énigme à résoudre.

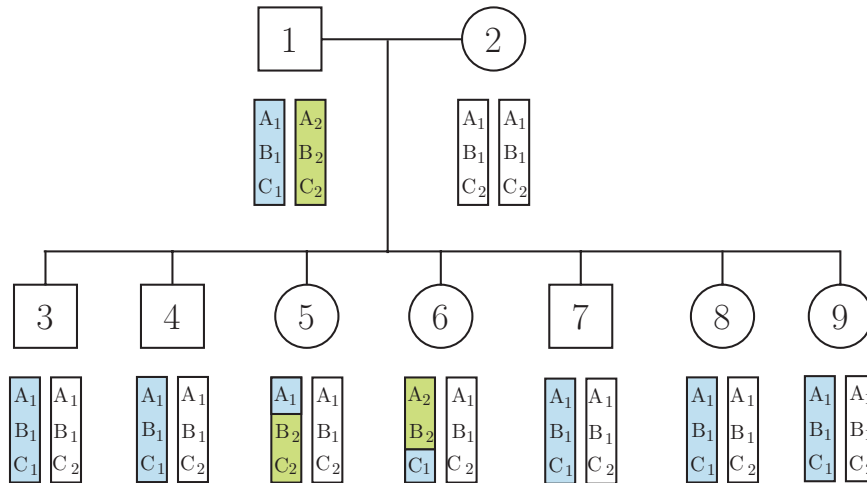


Figure 1.1.2. Illustration de la transmission des gènes dans une cellule familiale. Les carrés représentent les hommes, les cercles les femmes. Sous chacun des 9 individus est représentée une paire de chromosomes.

La figure 1.1.2 présente un schéma de la transmission des caractères génétiques dans une cellule familiale. Sous chaque individu ($i = 1, \dots, 9$), le génotype est présenté (le génotype réfère aux gènes, alors que l'action des gènes, ce qui est observable, est appelé phénotype). Les lettres sur chacun des chromosomes représentent des marqueurs génétiques. Un marqueur est une “trace” caractéristique de l'ADN que l'on peut situer le long du génome (c'est-à-dire à un locus spécifique) qui est polymorphique (c'est-à-dire il existe plusieurs “versions” de cette trace au

même endroit) dans la population que l'on étudie. Par exemple A_1 et A_2 sont deux différents allèles du marqueur A (cf. fig. 1.1.2). Il existe de nombreux types de marqueurs; les premiers étaient des phénotypes, comme le groupe sanguin par exemple. On travaille aujourd'hui essentiellement avec deux types de marqueurs: des microsatellites et des SNPs. Les microsatellites (cf. fig. 1.1.3 a) sont des répétitions d'une séquence de nucléotides: par exemple, à un certain endroit sur un chromosome donné, la séquence CA est répétée; chez certains individus, on trouve trois répétitions de ce code, chez d'autres cinq, etc. Un nouveau type de marqueur est apparu récemment: le Single Nucleotide Polymorphism (SNP) (cf. fig. 1.1.3 b). Ces marqueurs sont binaires et très nombreux dans le génome: leur nombre est estimé à environ 30 millions chez l'humain, c'est-à-dire 1 SNP toutes les 1000 paires de bases. Ils ont de plus un taux de mutation très faible de l'ordre de 10^{-6} à 10^{-8} par génération, comparativement à 10^{-2} à 10^{-5} pour les microsatellites. Dans cette thèse, nous commencerons par construire des modèles pour des SNPs, puis nous adapterons nos modèles aux microsatellites. Les SNPs ayant des taux de mutation moins élevés, ils sont probablement plus propices à la cartographie génétique, car les recombinaisons sont ainsi plus facilement détectables.

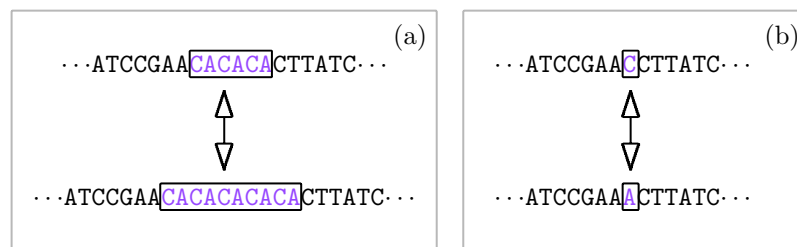


Figure 1.1.3. Illustration de l'évolution d'un marqueur:
(a) microsatellite, (b) SNP.

Le génotype de l'individu 1 de la figure 1.1.2 est constitué des deux haplotypes ' $A_1B_1C_1$ ' et ' $A_2B_2C_2$ '. Un haplotype est une suite ordonnée d'allèles de différents marqueurs. De façon générale, quand l'ADN d'une personne est génotypé (afin de connaître quels allèles elle possède à différents marqueurs), les haplotypes ne sont pas disponibles: ainsi on observerait que l'individu 1 possède les allèles A_1/A_2 au premier marqueur, B_1/B_2 au second, et C_1/C_2 au troisième, mais nous ne savons

pas si les haplotypes sont ‘ $A_1B_2C_2$ ’/‘ $A_2B_1C_1$ ’ ou ‘ $A_1B_1C_2$ ’/‘ $A_2B_2C_1$ ’ ou encore ‘ $A_1B_1C_1$ ’/‘ $A_2B_2C_2$ ’. Or cette information est précieuse, puisque de nombreuses méthodes de cartographie sont basées sur les informations des haplotypes. Les haplotypes peuvent être évalués par des méthodes de biologie moléculaire (encore très onéreuses), ou par l’utilisation de grands pedigrees (avec de l’information sur les grand-parents, par exemple). Dans les autres cas, des méthodes statistiques, telles que celles de Fallin et Schork (2000) ou de Stephens *et al.* (2001) peuvent être utilisées.

Pour cartographier un gène, l’événement biologique primordial est la recombinaison. S’il n’y avait pas de recombinaison, le père et la mère transmettraient chacun l’un des deux chromosomes de chaque paire, et les chromosomes de l’enfant seraient identiques aux chromosomes parentaux. En se référant à la figure 1.1.2, on peut voir que c’est le cas de l’enfant 7; par exemple, on voit que son chromosome paternel (de gauche) est ‘ $A_1B_1C_1$ ’ comme l’un des deux chromosomes de son père, et que son chromosome maternel est ‘ $A_1B_1C_2$ ’, mais on ne sait pas lequel des deux chromosomes de la mère a été transmis puisque la mère est homozygote aux trois marqueurs considérés (un individu est homozygote pour un certain gène s’il possède le même allèle sur chaque chromosome de la paire, sinon il est dit hétérozygote). Par contre, pour l’enfant numéro 5, on voit que son chromosome paternel est une composition des deux chromosomes de son père: il y a eu recombinaison entre le premier et le second marqueur. La recombinaison se produit pendant la division cellulaire, durant la formation des gamètes (ovules et spermatozoïdes). Deux chromosomes homologues échangent du matériel génétique lors d’un enjambement (“cross-over”, cf. fig. 1.1.4). Il faut un nombre impair d’enjambement pour qu’une recombinaison soit observable.

La probabilité que l’on ait une recombinaison entre deux marqueurs dépend de la distance entre ces marqueurs. Plus ils sont éloignés, plus le phénomène est probable. L’unité pour mesurer les distances entre loci est le Morgan. Un centiMorgan

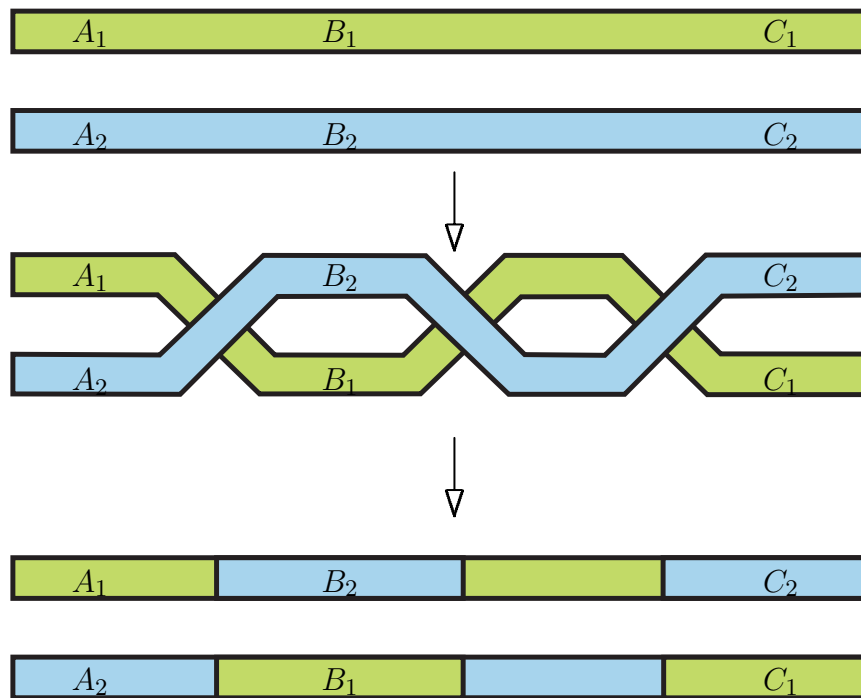


Figure 1.1.4. Nous observons trois marqueurs bialléliques A, B, et C ayant respectivement les allèles A_1/A_2 , B_1/B_2 et C_1/C_2 . Un enjambement se produit entre les marqueurs A et B, résultant en une recombinaison. Deux enjambements se produisent entre les marqueurs B et C, mais si nous observons seulement les trois marqueurs A, B et C nous ne pouvons pas constater les enjambements car les deux haplotypes obtenus sont ' $A_1B_2C_2$ ' et ' $A_2B_1C_1$ '.

(cM) correspond à la probabilité 1/100 d'avoir une recombinaison. Par exemple, si l'on se reporte à la figure 1.1.2, puisque nous observons 2 recombinants sur 7 entre le premier et dernier marqueurs, la distance génétique entre ces deux marqueurs est de $r = 2/7$ cM (centiMorgan). Dans la réalité, le phénomène de recombinaison est assez complexe: il ne varie pas seulement en fonction des distances; il existe des endroits où l'on a beaucoup de recombinaisons, d'autres où elles sont très rares. De plus, le taux de recombinaison varie selon le sexe. Enfin, la probabilité que l'on observe une recombinaison dans un intervalle donné n'est pas indépendante de la probabilité que l'on observe une recombinaison dans un intervalle voisin: c'est l'interférence. Considérons trois marqueurs A, B, et C, tel que r_{ij} soit la probabilité d'obtenir une recombinaison entre les marqueurs i et j ; la probabilité que l'on ait une double recombinaison, c'est-à-dire une recombinaison dans chaque

intervalle est $c \cdot r_{AB} \cdot r_{BC}$, où c est le coefficient de coïncidence. L'interférence est alors définie par $I = 1 - c$, et on a $I = 0$ s'il n'y a pas d'interférence. En pratique, la plupart des méthodes de cartographie supposent des séquences assez petites où la probabilité d'une double recombinaison est négligeable, et l'interférence peut alors être ignorée. Malheureusement, toute la complexité de la recombinaison s'avère difficile à modéliser; nous nous contenterons d'une approximation de la réalité.

1.2. Études par linkage et par déséquilibre de liaison

Pour localiser un gène mutant causant une maladie, deux approches sont possibles: le balayage du génome (“genome scan”) et l'utilisation de gènes candidats. Un balayage du génome ne suppose que très peu de connaissances préalables sur le gène causal: des centaines de marqueurs sont observés sur l'ensemble du génome, et l'on étudie l'association entre ces marqueurs et le phénotype. On s'attend à ce qu'une ou plusieurs régions montrent des signes évidents de liaison, et alors d'autres analyses plus fines sont effectuées afin de mieux localiser la mutation. L'autre approche consiste à utiliser les connaissances déjà acquises et à n'échantillonner que des marqueurs proches des gènes dont on sait *a priori* qu'ils pourraient jouer un rôle dans la détermination de la maladie. Pour chacune de ces deux approches, deux grandes méthodologies sont possibles: le linkage et le déséquilibre de liaison.

Les études de linkage utilisent des généalogies de familles (pédigrees) afin d'ordonner et de cartographier des gènes, avec comme but final de trouver celui ou ceux impliqués dans la maladie (Ott, 1999). La figure 1.2.1 illustre une généalogie hypothétique qui montre une maladie créée par l'allèle A_2 d'un gène à un locus donné. Si la généalogie est assez grande, ou si l'on dispose de plusieurs généalogies, on peut construire des modèles de transmission des chromosomes, en tenant évidemment compte des recombinaisons, afin de tester différentes hypothèses sur la position exacte du gène en question.

L'analyse de linkage est très utilisée pour des fins de cartographie. Elle présente

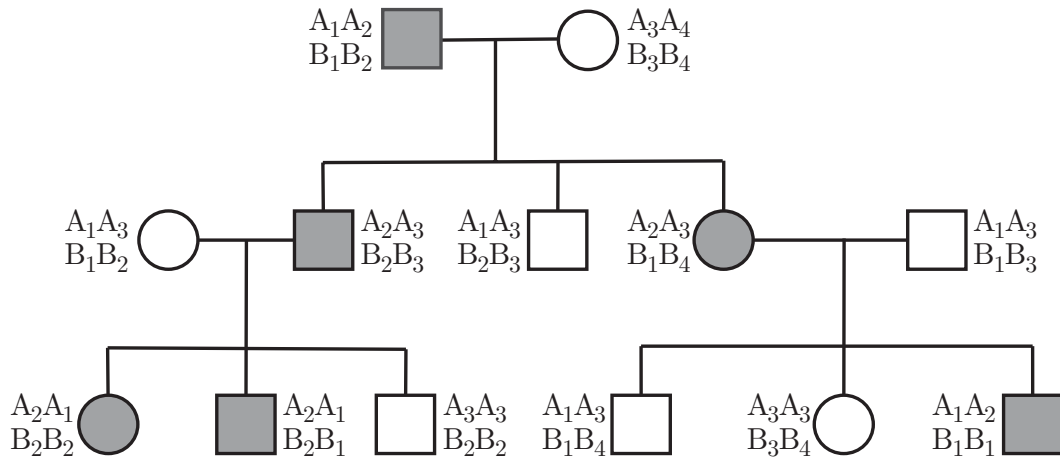


Figure 1.2.1. Généalogie illustrant la transmission d'un gène au locus A , ayant pour allèles A_1 , A_2 , A_3 et A_4 , dont l'allèle A_2 est responsable d'une maladie. Les individus atteints sont en gris; les hommes sont représentés par des carrés, les femmes par des cercles. Les allèles B_1 , B_2 , B_3 et B_4 représentent les allèles au locus B , lié au premier. Le matériel génétique paternel est à gauche, alors que le maternel est à droite de chaque génotype.

cependant un inconvénient majeur: sur l'échelle de quelques générations (il faut en effet connaître les généalogies pour procéder aux analyses), le nombre de recombinaisons est petit, et la résolution de l'estimateur est faible. C'est la raison pour laquelle les analyses par déséquilibre de liaison ont été développées. La figure 1.2.2 illustre l'évolution d'une séquence chromosomique au cours du temps. À un moment donné dans le passé ($t = t_0$), une mutation se produit sur l'un des chromosomes de la population; cette mutation crée une maladie, ou de façon plus générale, un caractère quelconque. Au fur et à mesure des différentes générations qui surviendront dans l'histoire du segment en question, il va y avoir mélange de matériel entre le chromosome sur lequel la mutation s'est produite et les autres chromosomes de la population. Plus l'histoire est longue, plus le mélange sera grand, c'est-à-dire moins le chromosome ayant subi la mutation sera conservé intact. Évidemment, on remarque que dans l'exemple de la figure 1.2.2, le chromosome mutant est sur-représenté au cours de l'évolution, ceci pour les besoins de l'exemple. On note qu'après plusieurs générations, à $t = t_2$, les cas (porteurs de la mutation) partagent un matériel génétique similaire (entre les lignes horizontales hachurées): c'est ce que l'on appelle le déséquilibre de liaison. Le déséquilibre de

liaison réfère à l'association non aléatoire d'allèles à des loci liés dans une population (Schork et Chakravarti, 1996). Considérons par exemple deux loci A et B liés, ayant chacun deux allèles 1 et 2, et notons A_1, A_2 les allèles du locus A , B_1, B_2 les allèles du locus B , et p_{ij} la fréquence de l'haplotype A_iB_j . Si, pour tout i et j , $p_{ij} = p_i p_j$, ou p_i (p_j) désigne la fréquence de l'allèle i (j) dans la population, alors on dit

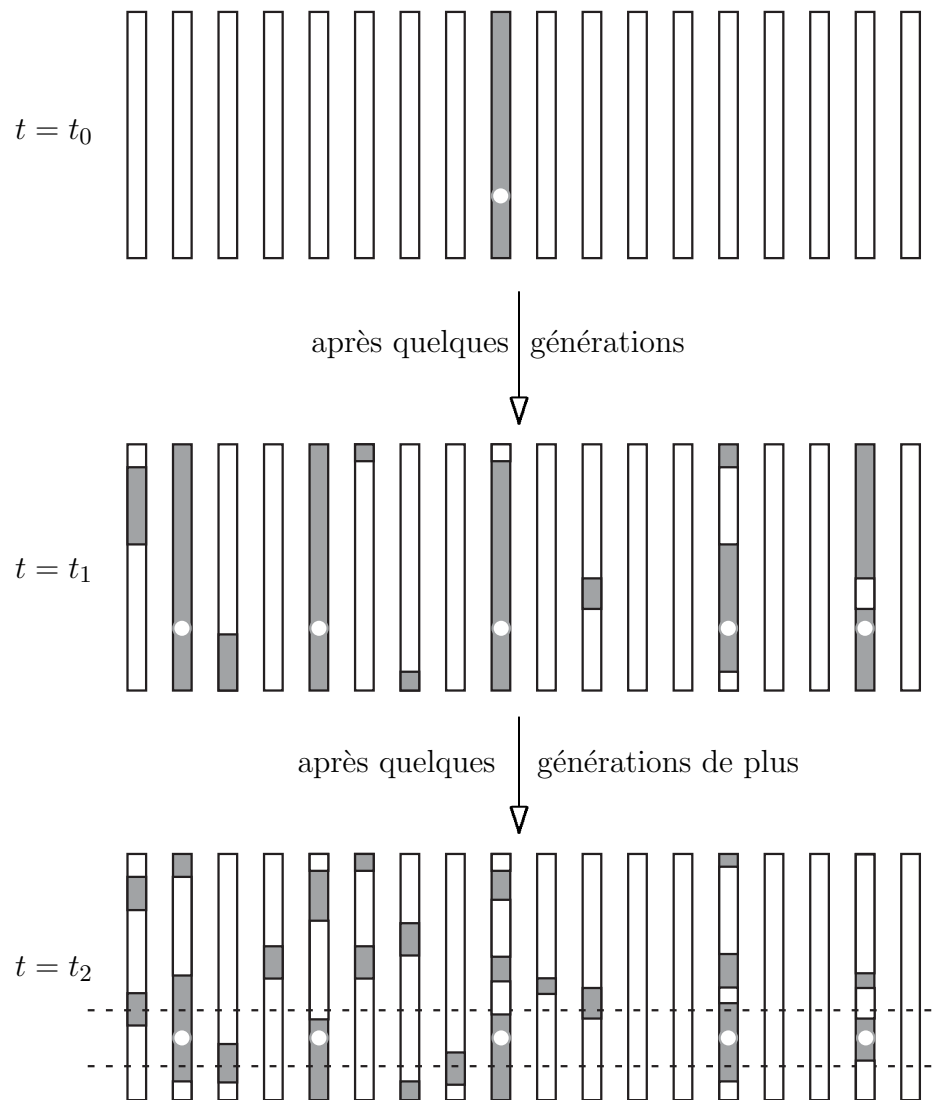


Figure 1.2.2. Illustration de l'évolution d'un segment chromosomique au cours du temps. À $t = t_0$, une mutation se produit sur l'un des chromosomes, et on peut suivre le partage chromosomique au cours du temps. Plus on avance dans le temps ($t_0 < t_1 < t_2$), plus les recombinaisons "cassent" le segment original et mélangent les séquences.

que les loci A et B sont en équilibre de liaison, sinon on a déséquilibre de liaison. Il existe plusieurs mesures de déséquilibre de liaison, qui sont décrites dans Hudson (2001a) par exemple.

Comme l'échelle de temps avec laquelle on travaille en déséquilibre de liaison est plus grande qu'en linkage, plus de recombinaisons se produiront, et l'on peut donc travailler sur des distances plus petites. Il est difficile, voire impossible d'avoir une résolution similaire avec des analyses de linkage à cause des tailles d'échantillon qui sont nécessaires (Boehnke, 1994). Si une population est récente, avec peu d'ancêtres fondateurs, elle est une candidate idéale pour des études utilisant le déséquilibre de liaison. De telles populations peuvent être trouvées au Québec, en Finlande, en Islande, ou chez certaines communautés fermées.

L'approche générale pour localiser une mutation causale en utilisant le déséquilibre de liaison est de prendre un échantillon de cas et de témoins, idéalement d'une population fondatrice (peu d'ancêtres), d'en obtenir des prises de sang, de chercher pour chaque individu la valeur d'un certain nombre de marqueurs et d'étudier l'association entre ces marqueurs et le phénotype. La façon la plus simple pour mesurer cette association est un test du χ^2 . Cependant, depuis quelques années, des méthodes de plus en plus sophistiquées apparaissent.

1.3. Maladies complexes

Les lois classiques de Mendel (1866) statuent que si le gène causant une maladie (ou un caractère de façon plus générale) est récessif, deux copies du gène mutant sont nécessaires pour que la maladie se déclare, alors que si le gène est dominant, une seule copie du gène mutant est suffisante. De tels gènes sont nombreux, et beaucoup d'entre eux ont déjà été identifiés, comme le gène causant la fibrose kystique par exemple. Un caractère génétique est appelé complexe à partir du moment où il ne suit pas les lois classiques de Mendel, et que plusieurs gènes sont impliqués dans la maladie (Lander et Schork, 1994, Ott, 1999). Pour une maladie complexe, l'association entre le phénotype et le génotype n'est plus

évidente. Cette association se trouve être compliquée par l'effet de plusieurs gènes, par des facteurs environnementaux, et par l'âge. Souvent, il y a alors pénétrance incomplète, un individu possède le gène mutant mais ne développe pas la maladie, et phénotopie, un individu développe la maladie mais ne possède pas le gène mutant. Le phénotype est encore dépendant du génotype, mais dans une moindre mesure. On parle d'hétérogénéité génétique si une mutation dans n'importe quel gène d'un ensemble de gènes cause le même phénotype. Enfin, on définit un caractère comme étant polygénique si celui-ci se manifeste par la présence de plusieurs gènes mutants. La statistique génétique commence à se pencher sur les maladies complexes, mais peu de modèles et méthodes sont encore disponibles.

Chapitre 2

De la cartographie génétique au processus de coalescence

Un aperçu des différentes méthodes de cartographie génétique fine va maintenant être présenté et nous étudierons en détail quelques unes de ces méthodes, afin d'illustrer la diversité des approches utilisées. Nous allons examiner les hypothèses utilisées, la construction de modèles décrivant l'évolution des séquences et les méthodes d'estimation. Cela nous amènera à décrire le processus de coalescence utilisé pour modéliser l'évolution des séquences, puis à son extension, le graphe de recombinaison ancestral qui introduit le phénomène de recombinaison dans le processus de coalescence.

2.1. Méthodes statistiques de cartographie génétique fine

La première approche pour faire de la cartographie fine est simple: utiliser un marqueur à la fois et faire un test d'association entre le marqueur et le phénotype.

Cette approche à l'avantage d'être rapide, mais s'est révélée plutôt inefficace: elle ignore toute la dépendance qui existe entre les marqueurs et la structure de population. Une toute première méthode utilisant plusieurs marqueurs à la fois fut celle de Terwilliger (1995), ce qui indique bien que ce type de méthodologie est très récent. Cependant, cette approche se contente d'utiliser une combinaison de statistiques qui sont construites à partir de paires de loci, comme d'ailleurs les méthodes de Xiong et Guo (1997), et celle de Collins et Morton (1998). Par contre, les méthodes de McPeck et Strahs (1999), de Morris *et al.* (2000), de Rannala et Reeve (2001), de Liu *et al.* (2001) et celle de Morris *et al.* (2002) modélisent l'haplotype complet. Un des problèmes majeurs est la modélisation de la structure de la population. L'arbre généalogique décrivant l'histoire de l'échantillon étant le plus souvent inconnu, sa structure doit être supposée ou inférée. Plusieurs méthodes supposent une généalogie en étoile (“star shape tree”) telle qu'illustrée à la figure 2.1.1. Cette structure suppose que les haplotypes observés descendent tous directement d'un seul ancêtre, de façon indépendante.

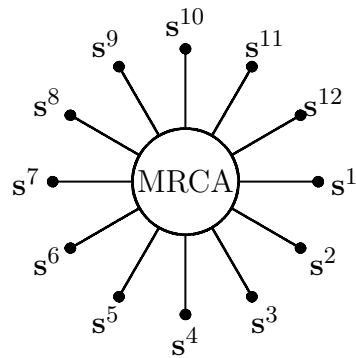


Figure 2.1.1. Généalogie en étoile: les 12 séquences s^1 à s^{12} descendent directement d'un ancêtre commun (le MRCA, pour “most recent common ancestor”).

Cette structure représente habituellement une mauvaise description des généalogies, mais elle peut cependant avoir un sens si la population montre un taux de croissance très rapide (Rannala et Slatkin, 1998; Slatkin et Hudson, 1991). C'est cette structure qui est utilisée par les méthodes de Terwilliger (1995), Xiong et Guo (1997), Collins et Morton (1998). L'arbre généalogique en étoile est simpliste, et, puisque l'histoire des séquences est en réalité commune, certains événements

peuvent être mal interprétés: par exemple, un segment partagé par plusieurs séquences peut être dû à un seul événement de recombinaison, ou un allèle mutant observé sur plusieurs séquences peut lui aussi être dû à un seul événement de mutation, ce qu'une généalogie en étoile ignore forcément, puisqu'elle va modéliser la probabilité que ce même événement se soit produit plusieurs fois. Cette description de généalogie peut ainsi conduire à des estimateurs de variance trop bas (Morris *et al.*, 2002).

Les méthodes de McPeck et Strahs (1999), Morris *et al.* (2000) et de Liu *et al.* (2001) utilisent également une structure de généalogie en étoile, mais ajustent la vraisemblance pour tenir compte de la corrélation entre les séquences. Pour la méthode de McPeck et Strahs (1999) par exemple, cela a pour effet d'obtenir des intervalles de confiance plus larges. Rannala et Slatkin (1998) utilisent un processus de coalescence intra-allélique (un processus de naissance et de mort) pour décrire la généalogie, tout comme Rannala et Reeves (2001). Graham et Thompson (1998) modélisent la généalogie par des arbres consistants avec les chromosomes cas, arbres générés selon un modèle de Moran (1962) dont les paramètres démographiques sont supposés connus. Lam *et al.* (2000) construisent un arbre généalogique des chromosomes cas en utilisant un mélange de parcimonie et de vraisemblance. Morris *et al.* (2002) utilisent quant à eux un processus de coalescence modifié, qui leur permet de modéliser plusieurs origines de la mutation au gène causal et la possibilité d'avoir une pénétrance incomplète.

2.1.1. Combinaison de statistiques à différents loci

Terwilliger (1995) propose une des premières méthodes d'analyse du déséquilibre de liaison. Nous présentons cette méthode parce qu'elle est simple, et qu'elle fut très utilisée. Afin d'étudier l'association entre une maladie et des marqueurs, Terwilliger introduit un paramètre λ ($0 \leq \lambda \leq 1$). La fréquence de l'allèle i dans la population est p_i , et nous noterons par j ($j = 1, \dots, m, j \neq i$) les autres allèles. Soit $q_i = p(i|D)$, la probabilité d'être porteur de l'allèle i si l'individu est atteint de la maladie; un cas est représenté par D , et un témoin par $+$. On pose:

$$P(i|D) = p_i + \lambda(1 - p_i),$$

$$P(j|D) = p_j - \lambda p_j \quad (i \neq j).$$

Le facteur λ permet de modéliser une augmentation de la fréquence allélique de i dans les chromosomes cas par rapport à la population en général; q_i prend alors des valeurs entre p_i et 1 (seules des associations positives sont possibles). Comme $p_i = P(i|D)P(D) + P(i|+)P(+)$, on a:

$$\begin{aligned} P(i|+) &= \frac{1}{P(+)} [p_i - P(i|D)P(D)] \\ &= \frac{1}{P(+)} [p_i(1 - P(D)) - \lambda(1 - p_i)P(D)] \\ &= p_i - \lambda(1 - p_i)P(D)/(1 - P(D)). \end{aligned}$$

De même, on a:

$$P(j|+) = p_j + \lambda p_j P(D)/(1 - P(D)).$$

La vraisemblance est donc:

$$L_i = P(i|D)^{x_i} P(i|+)^{y_i} \prod_{j=1(i \neq j)}^m P(j|D)^{x_j} P(j|+)^{y_j},$$

où x_i, y_i, x_j et y_j sont les effectifs observés. La vraisemblance en considérant tous les allèles est:

$$L = \sum_{i=1}^m p_i L_i.$$

La vraisemblance est alors maximisée par rapport à λ . Un test du rapport de vraisemblance est alors utilisé pour déterminer si le paramètre λ est différent de 0, ce qui impliquerait alors que l'on ait déséquilibre de liaison entre le marqueur et le gène causant la maladie. Terwilliger propose ensuite une extension pour tenir compte du fait que plusieurs allèles puissent être associés à la maladie en paramétrant le modèle avec plusieurs λ : λ_i pour l'allèle i , λ_{i+1} pour l'allèle $i + 1$, etc., puis une dernière extension pour estimer la position du gène en paramétrant le modèle par $\lambda = \alpha(1 - \theta)^g$, où θ est la distance entre le marqueur et la mutation

cherchée, et g est le nombre de générations depuis l'introduction de la mutation causant la maladie.

2.1.2. Entre parcimonie et vraisemblance

La méthode de Lam *et al.* (2000) est intéressante à plus d'un titre. Nous allons voir que bien que l'approche soit assez différente de ce que l'on propose dans cette thèse, les deux méthodes partagent plusieurs points communs.

Les données sont un échantillon d'haplotypes de cas et de témoins. On suppose que la mutation causale se situe entre deux marqueurs spécifiés connus et qu'elle seule est responsable de tous les haplotypes de cas. La méthode se présente comme une combinaison de parcimonie et de vraisemblance pour construire un arbre d'évolution des haplotypes, avec chaque nœud séparé par une mutation (Mu), ou par une recombinaison (Re) de son nœud parent. Le principe conducteur pour connecter les nœuds est que la mutation causale doit être conservée dans l'intervalle supposé pour tous les haplotypes. Si plus d'un événement de Mu ou de Re sont nécessaires pour connecter deux nœuds, des nœuds latents sont insérés pour compléter l'arbre (cf. fig. 2.1.2). Une fois l'arbre construit, sa vraisemblance est calculée à partir d'un modèle de probabilité et de l'information *a priori* disponible. Sur la base d'un modèle de vraisemblance, chaque intervalle a une probabilité *a posteriori* que la mutation causale soit dans cet intervalle.

Pour le cas hétérogène, quand une seule mutation n'est pas responsable de tous les haplotypes malades, seulement des sous-ensembles d'haplotypes peuvent être organisés en arbres cohérents, et le nombre de ces groupes est inconnu. Pour chaque position de la mutation et d'un nombre de groupes spécifiés, l'algorithme organise les données en groupes homogènes et construit un arbre pour chaque groupe; les inférences sont basées sur le plus grand groupe seulement. La méthode est basée sur la vraisemblance, mais n'essaye pas d'évaluer une "pleine vraisemblance". La simplicité a été choisie pour faciliter l'interprétation, réduire le nombre d'hypothèses et accroître la flexibilité et l'extensibilité du modèle.

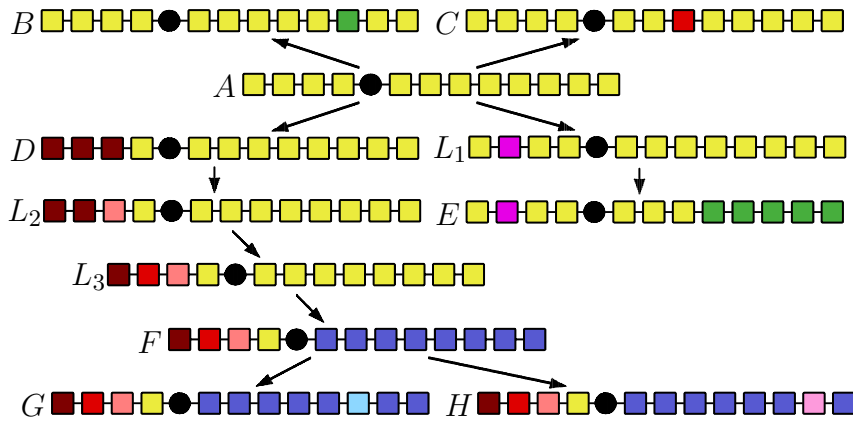


Figure 2.1.2. Les blocs représentent des marqueurs, et les couleurs les allèles de ces marqueurs. La mutation (●) est supposée être dans l'intervalle 4, entre les marqueurs 4 et 5. A est la racine. Par exemple, $A \rightarrow D$ et $L_1 \rightarrow E$ sont des recombinaisons, et les connexions $A \rightarrow B$, $A \rightarrow C$, $A \rightarrow L_1$ se font par des événements de mutation ou de recombinaison. Comme il est impossible de connecter A et E par un simple événement, un haplotype latent L_1 est ajouté à l'arbre (Adaptée de Lam *et al.*, 2000).

Un haplotype est une suite de marqueurs ordonnés (cf. fig. 2.1.3), de gauche à droite, étiquetés de 1 à L . L'intervalle contenant la mutation est d ($d = 1, \dots, L - 1$), et D est l'ensemble des intervalles considérés. Soit \mathcal{T} l'espace des topologies d'arbres qui décrivent les liens entre les haplotypes, et soit τ une topologie particulière. Le paramètre d'intérêt est $\lambda = (d, \tau)$. Soit Y un échantillon et soit $Y' = (S_0, \dots, S_n)$ la série unique des haplotypes dans Y avançant dans le temps, de la racine S_0 jusqu'aux branches, comprenant les haplotypes latents.

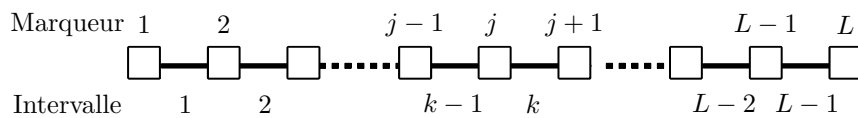


Figure 2.1.3. Illustration d'une séquence, avec L marqueurs et $L - 1$ intervalles.

Sachant λ , la vraisemblance de Y est, par les propriétés markoviennes de tels arbres:

$$L(d, \tau) = P(Y|d, \tau) = P_\lambda(S_0 \text{ est racine}) \times \prod P_\lambda(S_j|S_i), \quad (2.1)$$

où le produit est pris sur toutes les arêtes de l'arbre. Ici, $T|S$ signifie que T est relié directement à S par un événement de Re ou de Mu , et on a :

$$P_\lambda(T|S) = P_\lambda(T|S, Mu)P(Mu|Mu \text{ ou } Re) + P_\lambda(T|S, Re)P(Re|Mu \text{ ou } Re). \quad (2.2)$$

Si S et T diffèrent à plus d'un marqueur, alors $P_\lambda(T|S, Mu) = 0$, et S et T sont reliés par un événement de recombinaison; s'ils diffèrent à un marqueur seulement, les deux termes contribuent à (2.2). Soit γ_j la probabilité d'une mutation au marqueur j , et θ_k est la probabilité d'une recombinaison dans l'intervalle k . La probabilité totale d'une seule mutation est $A = \sum_j \gamma_j \prod_{i \neq j} (1 - \gamma_i)$ et celle d'une seule recombinaison est $B = \sum_k \theta_k \prod_{i \neq k} (1 - \theta_k)$. D'autre part, $P_\lambda(S_0 \text{ est racine})$ est estimé par la proportion des haplotypes S_0 dans Y , sachant que les haplotypes de cas plus anciens sont susceptibles d'être plus fréquents.

Une paire d'haplotypes sont identiques par état (IBS: "Identical by state") s'ils partagent les mêmes allèles à tous les marqueurs, quelle que soit leur relation de descendance; ils sont identiques par descendance (IBD: "Identical by descent") s'ils sont IBS et que le matériel génétique de l'un provient de l'autre par descendance. De plus, Lam *et al.* (2000) définissent IBS* comme étant IBS mais partageant au plus une portion de l'haplotype IBD. Supposons que S et T apparaissent reliés par un événement de Re , qu'ils sont IBS à gauche de k_0 ($k_0 \geq d$) et diffèrent à droite de k_0 . Soit R l'intervalle où la recombinaison a lieu. Le point de recombinaison peut être n'importe où entre d et k_0 , et on a donc :

$$P_\lambda(T, S, Re) = \sum_{k=d}^{k_0} P_\lambda(T|S, Re, R = k)P(R = k).$$

Soit $S^{m:n}$ le fragment de S couvert par les marqueurs $m, m + 1, \dots, n$. Si T est directement relié à S par une cassure de recombinaison dans l'intervalle k , alors $T^{1:k}$ doit être IBD à $S^{1:k}$, et $T^{(k+1):L}$ est un haplotype partiel obtenu par recombinaison. Donc, on a :

$$\begin{aligned} P_\lambda(T|S, Re, R = k) &= P_\lambda(T^{1:k} \text{ IBD à } S^{1:k}, T^{(k+1):L}|T^{1:k} \text{ IBS à } S^{1:k}) \\ &= P_\lambda(T^{1:k} \text{ IBD à } S^{1:k}|T^{1:k} \text{ IBS à } S^{1:k})P_\lambda(T^{(k+1):L}). \end{aligned}$$

De plus, on a :

$$\begin{aligned} &P_\lambda(T^{1:k} \text{ IBD à } S^{1:k}|T^{1:k} \text{ IBS à } S^{1:k}) \\ &= \frac{P_\lambda(T^{1:k} \text{ IBD à } S^{1:k})}{P_\lambda(T^{1:k} \text{ IBD à } S^{1:k}) + P_\lambda(T^{1:k} \text{ IBS}^* \text{ à } S^{1:k}) \times P_\lambda(T^{1:k}|T^{1:k} \text{ IBS}^* \text{ à } S^{1:k})}, \end{aligned}$$

car les haplotypes partiels $T^{1:k}$ et $S^{1:k}$ sont IBS s'ils sont IBD, ou si seulement une partie de la séquence est IBD. Il reste alors à calculer $P_\lambda(T^{(k+1):L})$. D'autre part, la probabilité d'une recombinaison dans une séquence est $\theta = \sum_{k=1}^{L-1} \theta_k$, et alors :

$$P(R = k) \equiv P(R = k | 1 \text{ recombinaison}) = \frac{(1 - \theta)^g \theta_k}{(1 - \theta)^g \theta} = \frac{\theta_k}{\theta},$$

où $g + 1$ est le nombre de générations jusqu'à ce que S recombine pour former T . Lam *et al.* (2000) assument également que :

$$P_\lambda(T^{1:k} \text{ IBD à } S^{1:k}) = P_\lambda(T^{1:k} \text{ IBS}^* \text{ à } S^{1:k}) = \frac{1}{2}.$$

Il s'agit d'une probabilité non informative qui ne devrait favoriser aucune des hypothèses.

Si deux haplotypes S et T sont reliés par un événement de mutation, ils diffèrent exactement à un marqueur que nous noterons M . Soit S^{-m} l'haplotype S excluant m . Soit $\Delta_m = |S^{m:m} - T^{m:m}|$ la taille de la mutation, i.e. la variation du nombre de répétitions du microsatellite, et supposons que Δ_m a une distribution de probabilité μ_m . Si S et T diffèrent à la position $M = m_0$, alors la vraisemblance d'une mutation est $P_\lambda(T|S, Mu) = P_\lambda(T|S, Mu, M = m_0)P(M = m_0)$. Si T et S sont reliés par un événement de mutation à la position m_0 , alors T^{-m_0} est IBS à S^{-m_0} , et $T^{m_0:m_0}$ est obtenu par une mutation de taille δ_{m_0} . Donc :

$$\begin{aligned}
P_\lambda(T|S, Mu, M = m_0) &= P_\lambda(T^{-m_0} \text{ IBD à } S^{-m_0}, \Delta_{m_0} = \delta_{m_0} | T^{-m_0} \text{ IBS à } S^{-m_0}) \\
&= P_\lambda(T^{-m_0} \text{ IBD à } S^{-m_0} | T^{-m_0} \text{ IBS à } S^{-m_0}) P_\lambda(\Delta_{m_0} = \delta_{m_0}).
\end{aligned}$$

En pratique, les taux de mutation et la distribution des changements dûs à la mutation sont nécessaires. Si les marqueurs considérés sont des microsatellites, le taux de mutation est entre 10^{-2} et 10^{-5} ; s'il s'agit de SNPs, entre 10^{-6} et 10^{-8} .

Comme on l'a vu précédemment, des estimés de $P_\lambda(T^{m:n})$, la probabilité des haplotypes partiels, sont nécessaires. On suppose qu'un échantillon d'haplotypes est disponible, duquel les fréquences des haplotypes partiels peuvent être estimées. Si l'échantillon de référence est grand, cette probabilité peut être estimée par la fréquence de $T^{m:n}$ dans l'échantillon. Cependant, cette fréquence peut être nulle et doit alors être estimée d'une autre façon. La probabilité d'observer l'haplotype partiel $[x_1 x_2 \dots x_m]$ peut s'écrire:

$$P(x_1 x_2 \dots x_m) = P(x_1) P(x_2 | x_1) P(x_3 | x_1 x_2) \dots P(x_m | x_1 x_2 \dots x_{m-1}).$$

Une approche pour estimer cette quantité est de supposer l'indépendance des marqueurs et de faire le produit des fréquences d'allèles. Pour tenir compte des dépendances entre allèles, les probabilités conditionnelles peuvent être basées sur le plus haut niveau de dépendance des haplotypes dans l'échantillon de référence. Supposons que l'haplotype le plus long ressemblant à $[x_1 x_2 \dots x_j]$ et incluant x_j est $[x_{j-2} x_{j-1} x_j]$. Alors:

$$P(x_j | x_1 x_2 \dots x_{j-1}) = P(x_j | x_{j-2} x_{j-1}) = \frac{P(x_{j-2} x_{j-1} x_j)}{P(x_{j-2} x_{j-1})}.$$

Finalement, on cherche \hat{d} en trouvant $\hat{\lambda} = (\hat{d}, \hat{\tau})$, tel que:

$$\mathcal{L}(\hat{d}, \hat{\tau}) = \max_{d \in \mathcal{D}, \tau \in \mathcal{T}} \mathcal{L}(d, \tau) = \max_{d \in \mathcal{D}, \tau \in \mathcal{T}} P(Y | d, \tau).$$

Aussi, une analyse bayésienne basée sur la distribution marginale *a posteriori* de d donne:

$$\begin{aligned} P(d|Y) &\propto \pi_{\mathcal{D}}(d) \sum_{\tau \in \mathcal{T}} P(Y|d, \tau) \pi_{\mathcal{T}}(\tau) \\ &= \pi_{\mathcal{D}}(d) P(Y|d). \end{aligned}$$

Un estimé de d est alors le mode de la distribution *a posteriori*. Notons que l'expression précédente n'est pas évaluable en pratique. Les inférences faites sur d demandent une maximisation sur l'espace des arbres \mathcal{T} , de cardinalité infinie. Une solution consiste à construire de façon heuristique un arbre que l'on pense proche de l'arbre optimal. La méthode de construction choisie favorise un minimum de nœuds inobservés (parcimonie). Le processus est itératif, commençant par regrouper les haplotypes les plus proches puis en regroupant ces groupes.

La méthode passe par la modélisation d'un haplotype ancestral, c'est-à-dire un unique haplotype ancestral, mais il peut exister dans la réalité plusieurs haplotypes ancestraux. Ceci peut être causé par plusieurs raisons: (1) Un seul haplotype ancestral mais on a perdu le début de l'histoire; (2) la mutation s'est produite plusieurs fois sur plusieurs chromosomes ancestraux; (3) plus d'une mutation est présente dans le gène à l'étude; (4) des individus malades ne possèdent pas la mutation. La solution consiste à trouver un groupe d'haplotypes homogènes. La solution proposée par les auteurs consiste à chercher différents groupes d'haplotypes homogènes, mais de ne faire l'analyse que sur le plus grand groupe, qui devrait être celui qui contient les haplotypes descendant de l'unique haplotype ancestral.

Les auteurs soulignent que la méthode n'est pas robuste à des erreurs d'ordre de marqueurs, mais toutes les méthodes multilocus souffrent de ce problème. En effet, bien que les cartes génétiques et les estimations des taux de recombinaison entre marqueurs s'améliorent constamment, on peut croire que la séquence ordonnée des marqueurs X, Y, et Z est X-Y-Z, alors qu'en réalité la séquence est X-Z-Y.

2.1.3. Décroissement du partage d'haplotypes (DHS)

McPeck et Strahs (1999) proposent une méthode de cartographie fine basée sur la décroissance du partage d'haplotypes [†], appelée DHS (Decay of Haplotype Sharing). On modélise la distribution de l'étendue du partage d'haplotypes autour d'une certaine mutation.

Supposons que la mutation causale se situe au locus 0, avec les loci 1, 2, 3, ... à distance croissante d'un côté de 0, et les loci -1, -2, -3, ... à distance croissante de l'autre côté de 0 (cf. fig. 2.1.4). Soit x_i la distance en Morgan du locus i au locus 0, et soit $d_{j,i} = x_i - x_j$ pour $j < i$. Supposons que l'haplotype h_{obs} , un descendant de $\tau^{\text{ème}}$ génération, hérite de la mutation causale, ainsi que d'un bloc intact, entre les loci $-k$ et j inclusivement, mais que cet haplotype ne soit plus intact à l'extérieur de ce bloc, comme illustré dans la figure 2.1.4.

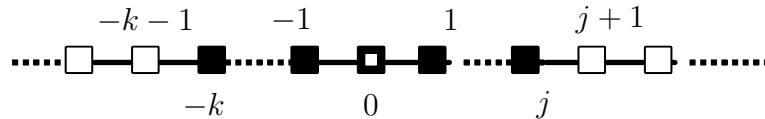


Figure 2.1.4. Illustration du modèle: l'haplotype h_{obs} est ancestral autour de la mutation (■, locus 0), du locus $-k$ jusqu'au locus j . Un locus ancestral est représenté par ■ et un locus non ancestral par □.

Si l'on suppose qu'il n'y a ni sélection ni interférence, alors les distances génétiques entre le locus 0 et les points de cassure $-k$ et j chez le descendant de $\tau^{\text{ème}}$ génération sont distribués comme des variables aléatoires exponentielles de taux τ (McPeck et Strahs, 1999). La longueur du segment ancestral hérité (en Morgan) a alors une distribution gamma de paramètres 2 et τ . De plus, la longueur moyenne du segment ancestral est $2/\tau$ Morgan, et la distance moyenne de la mutation à chacun des points de cassure du bloc ancestral est τ^{-1} Morgan. Le paramètre τ^{-1} est donc une mesure naturelle du déséquilibre de liaison. De plus, la distribution de τ^{-1} dans de petits échantillons se rapproche

[†] cf. section 1.1 page 8 pour la définition d'un haplotype

d'une distribution normale, ce qui facilite la construction des intervalles de confiance. La vraisemblance de l'haplotype h_{obs} est la probabilité qu'il n'y ait pas d'enjambement ("cross-over") entre les loci $-k$ et j , qu'il y ait au moins un enjambement entre les loci $-k - 1$ et $-k$, et au moins un autre entre les loci j et $j + 1$, tout cela durant τ générations, c'est-à-dire:

$$\begin{aligned} L(\tau^{-1}, h_{obs}) &\propto g(\tau^{-1}, -k, j) \\ &= e^{-\tau d_{-k,j}}(1 - e^{-\tau d_{-k-1,-k}})(1 - e^{-\tau d_{j,j+1}}), \end{aligned}$$

en assumant que les marqueurs soient assez proches pour qu'un double enjambement (inobservable) entre deux marqueurs contigus est impossible.

Pour un haplotype de $\tau^{\text{ème}}$ génération, considérons la fonction $R(x) = A$ (pour ancestral) aux positions x dans le plus grand bloc intact autour de la mutation, et $R(x) = N$ (pour non ancestral) aux autres positions. Alors $R(x)$ est une double chaîne de Markov indexée par la position de la mutation et allant dans deux directions opposées:

$$\begin{aligned} P[R(x+t) = A | R(x) = A] &= \exp(-\tau|t|), \\ P[R(x+t) = N | R(x) = A] &= 1 - \exp(-\tau|t|), \\ P[R(x+t) = N | R(x) = N] &= 1 \end{aligned}$$

pour x et t tous deux positifs, ou tous deux négatifs. L'haplotype ancestral (h_{anc}) n'est cependant pas connu. Il faut donc tenir compte dans la vraisemblance du fait que des allèles soient observés, sachant qu'ils sont non ancestraux. La vraisemblance de l'haplotype observé (h_{obs}) s'écrit alors:

$$\begin{aligned} L(h_{anc}, \tau^{-1}; h_{obs}) &= g(\tau^{-1}, -k, j) \\ &\quad \times P_{null}[h_{obs}(j+1), h_{obs}(j+2), \dots] \\ &\quad \times P_{null}[h_{obs}(-k-1), h_{obs}(-k-2), \dots], \end{aligned} \tag{2.3}$$

où $P_{null}[h_{obs}(x+1), h_{obs}(x+2), \dots]$ est la probabilité conjointe que les allèles aux positions $x+1, x+2, \dots$ de l'haplotype h_{obs} soient non ancestraux. On

utilise alors les données sur les contrôles pour estimer cette quantité. Si seulement les fréquences des allèles sont disponibles chez les contrôles, $P_{null}[h_{obs}(j+1), h_{obs}(j+2), \dots]$ peut être estimé par $\prod_{l \geq j+1} f[h_{obs}(l)]$, où $f[h_{obs}(l)]$ est la fréquence de l'allèle $h_{obs}(l)$ chez les contrôles. Si des haplotypes contrôles sont disponibles, une chaîne de Markov d'ordre k peut être utilisée, mais il n'y a en général pas assez d'information pour utiliser un ordre $k > 1$. On estime alors $P_{null}[h_{obs}(j+1), h_{obs}(j+2), \dots]$ par le produit des fréquences conditionnelles $f[h_{obs}(l+1)|h_{obs}(l)]$, défini par:

$$f[h_{obs}(l+1)|h_{obs}(l)] = \frac{\text{effectif de la paire } h_{obs}(l+1)|h_{obs}(l)}{\text{effectif de } h_{obs}(l)},$$

où les fréquences sont calculées dans la population des contrôles. Si un estimé de fréquence est nul, on l'ajuste en lui assignant une petite valeur. La vraisemblance (2.3) est alors maximisée en fonction de h_{anc} et de τ^{-1} simultanément.

Jusqu'ici, on a supposé que le bloc partagé par h_{obs} et h_{anc} est identique par descendance, mais ils peuvent partager une partie de ce bloc seulement par chance (à cause de recombinaisons): on ajuste la vraisemblance pour que le premier point de recombinaison de gauche puisse être entre $-k-1$ et 0, et le premier point de recombinaison de droite entre 0 et $j+1$, et l'on obtient:

$$\begin{aligned} L(h_{anc}, \tau^{-1}; h_{obs}) &= \sum_{i=0}^j \sum_{l=0}^k g(\tau^{-1}, -l, i) \\ &\quad \times P_{null}[h_{obs}(i+1), h_{obs}(i+2), \dots] \\ &\quad \times P_{null}[h_{obs}(-l-1), h_{obs}(-l-2), \dots]. \end{aligned}$$

Toutes les possibilités sont alors permises: h_{obs} peut partager n'importe quel bloc avec h_{anc} , soit parce qu'il est réellement ancestral, soit par chance.

Soit P_l la matrice de transition au locus l lorsqu'il y a mutation, dans laquelle l'entrée (i, j) est la probabilité de passer de l'allèle i à l'allèle j . La probabilité qu'un haplotype de $\tau^{\text{ème}}$ génération ait l'allèle j au locus l sachant que l'haplotype ancestral a l'allèle i au même locus et qu'il n'y ait pas eu d'enjambement entre les loci 0 et l durant les τ générations est alors:

$$m(l, \tau, i, j) = \sum_{k=0}^{\tau} \binom{\tau}{k} m_l^k (1 - m_l)^{\tau-k} (P_l^k)_{ij},$$

où m_l est le taux de mutation par génération au locus l . Pour simplifier, on suppose que $m_l = m$ pour tout l , $(P_l)_{ij} = 0$ si $i = j$, et $(P_l)_{ij} = \frac{1}{(n_l-1)}$ sinon, où n_l est le nombre d'allèles au locus l ; cela veut dire que l'on suppose que le taux de mutation est le même pour tous les marqueurs et que, quand une mutation se produit, la mutation se fait au hasard vers n'importe lequel des autres allèles. Alors, on a:

$$m(l, \tau, i, i) = \left(1 - m \frac{n_l}{n_l - 1}\right)^{\tau} + \left[1 - \left(1 - m \frac{n_l}{n_l - 1}\right)^{\tau}\right] \frac{1}{n_l},$$

et $m(l, \tau, i, j) = [1 - m(l, \tau, i, i)] / (n_l - 1)$ si $i \neq j$. L'introduction de mutations complique le calcul de la vraisemblance: un bloc de h_{obs} ancestral jusqu'au locus j , peut être en fait ancestral jusqu'au locus $j + 3$, par exemple, et n'être non ancestral qu'au locus $j + 1$ par une mutation. Soient l_{re} et $-l_{le}$ les indices des marqueurs à droite et à gauche respectivement de la séquence. On introduit donc la probabilité de mutation à chaque locus dans la vraisemblance, et celle-ci s'écrit alors:

$$\begin{aligned} L(h_{anc}, \tau^{-1}; h_{obs}) &= \sum_{i=0}^{l_{re}} \sum_{l=0}^{l_{le}} \left[g(\tau^{-1}, -l, i) \prod_{k=-l}^i m(k, \tau, h_{anc}(k), h_{obs}(k)) \right. \\ &\quad \times P_{null}[h_{obs}(i+1), h_{obs}(i+2), \dots] \\ &\quad \left. \times P_{null}[h_{obs}(-l-1), h_{obs}(-l-2), \dots] \right]. \end{aligned} \quad (2.4)$$

Après avoir défini la vraisemblance d'un individu, il reste à définir celle de l'échantillon. Dans un premier temps, les auteurs supposent l'indépendance des observations, et la vraisemblance est obtenue par le produit des vraisemblances individuelles. Cela correspond au cas où l'on aurait une histoire généalogique en étoile, c'est-à-dire que chaque séquence descend directement de l'ancêtre. Le calcul de la vraisemblance (2.4) implique la sommation, pour chaque séquence, de probabilités sur tous les enjambements, sur un grand nombre d'ancêtres, etc,

ce qui est compliqué. Les auteurs utilisent plutôt la structure markovienne de la vraisemblance, et emploient l'algorithme de Baum/EM (Baum, 1972) pour la maximiser.

Tenir compte de la dépendance entre individus est extrêmement compliqué; c'est pourquoi les auteurs utilisent les covariances entre observations et maximisent la quasi-vraisemblance. Ils montrent que la quasi-vraisemblance est la même pour les deux cas (dépendance et indépendance des observations), mais que l'erreur standard doit être ajustée.

La méthode de McPeck et Strahs (1999) est très élégante. En se basant sur une généalogie en étoile, ils modélisent d'une façon tout à fait intéressante le décroissement du partage d'haplotypes. D'après les exemples fournis par les auteurs, la méthode donne de bonnes estimations, mais accompagnées d'intervalles de confiance un peu grands.

2.1.4. Reconstruction d'haplotypes ancestraux (AHR)

Service *et al.* (1999) présentent une méthode d'étude du déséquilibre de liaison à des fins de cartographie à l'échelle du génome. C'est une méthode pour plusieurs loci, mais qui reste, comme nous allons le voir, très simple. Cette méthode, nommée AHR (Ancestral Haplotype Reconstruction), compare la distribution des haplotypes chez des individus affectés versus ce qui est attendu chez des individus descendus d'un ancêtre commun qui portait une mutation du gène causal.

La méthode utilise trois paramètres: la position du gène causal (x), la proportion des haplotypes cas (c'est-à-dire porteurs du gène mutant) descendant de l'ancêtre commun (porteur du gène mutant) (α), et le nombre de générations jusqu'à l'ancêtre commun (g). La vraisemblance sera maximisée sur x , g , et α puis comparée à la vraisemblance sous l'hypothèse nulle (H_0), quand $\alpha = 0$, c'est-à-dire lorsque aucun cas de l'échantillon ne descend de l'ancêtre. Considérons un chromosome avec trois marqueurs A , B et C , où la mutation causant la maladie

(D) se trouve entre les marqueurs A et B , (cf. fig. 2.1.5).

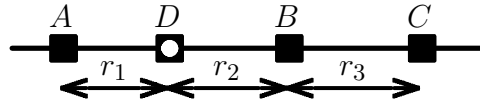


Figure 2.1.5. Illustration de la mutation causant la maladie D , avec les trois marqueurs A , B , et C .

Soient r_1 , r_2 et r_3 les fractions de recombinaison entre A et D , D et B , et B et C , respectivement. Soient de plus g le nombre de générations depuis que la mutation a été introduite dans la population, c'est-à-dire le temps jusqu'à l'ancêtre commun, et f_{x_j} la fréquence de l'allèle x_j du marqueur j . La probabilité qu'un individu affecté, donc porteur de la mutation originale, ait encore les mêmes allèles x_A , x_B , et x_C aux marqueurs A , B , C après g générations est:

$$\begin{aligned}
 & (1 - r_1)^g (1 - r_2)^g (1 - r_3)^g \\
 & + (1 - r_1)^g (1 - r_2)^g [1 - (1 - r_3)^g] f(x_C) \\
 & + (1 - r_1)^g [1 - (1 - r_2)^g] f(x_B) f(x_C) \\
 & + [1 - (1 - r_1)^g] (1 - r_2)^g (1 - r_3)^g f(x_A) \\
 & + [1 - (1 - r_1)^g] (1 - r_2)^g [1 - (1 - r_3)^g] f(x_A) f(x_C) \\
 & + [1 - (1 - r_1)^g] [1 - (1 - r_2)^g] f(x_A) f(x_B) f(x_C),
 \end{aligned} \tag{2.5}$$

c'est-à-dire, la probabilité qu'il n'y ait pas eu de recombinaison dans chacun des intervalles, ou qu'il n'y ait pas eu de recombinaison dans les deux premiers intervalles et qu'il y ait eu au moins une recombinaison dans le troisième intervalle et l'allèle au marqueur C est x_C avec probabilité $f(x_C)$, et ainsi de suite. Il s'agit ici en fait d'une énumération de tous les cas possibles. Notons que l'on permet une identité par état plutôt qu'une identité par descendance. Les probabilités d'autres haplotypes se calculent de façon similaire. On suppose qu'il n'y a pas d'interférence[†], et que la distribution des allèles des marqueurs est la même aujourd'hui que g générations plus tôt, dans les mêmes fréquences (la population est

[†] cf. section 1.1 page 10 pour la définition d'interférence

en équilibre de liaison). Les estimés des fréquences des allèles des différents marqueurs sont traités comme des paramètres de nuisance, et sont obtenus séparément sous H_0 et sous H_1 .

Supposons n_j allèles au marqueur j . Il y a alors $H = n_A \times n_B \times n_C$ haplotypes possibles. Soit T la matrice de transition de ces H haplotypes, tel que T_{ij} est la probabilité que l'haplotype ancestral i conduise à l'haplotype j en g générations.

La procédure peut être modifiée pour tenir compte d'un facteur d'hétérogénéité α , représentant la fraction des chromosomes descendant vraiment de l'ancêtre commun. Si la matrice calculée sous H_1 (le locus causal est entre A et B) est T , et la matrice calculée sous H_0 (équilibre de liaison) est T_n , alors une nouvelle matrice de transition pour une valeur particulière du paramètre α est:

$$T^* = \alpha T + (1 - \alpha)T_n.$$

Une fois que les probabilités de transition ont été estimées (par 2.5), on calcule la vraisemblance qu'un haplotype ancestral particulier produise l'échantillon d'haplotypes observés. Si la mutation survient sur un chromosome d'haplotype K , alors la vraisemblance que ce chromosome soit l'ancêtre des chromosomes observés est:

$$L_K = \prod_{i=1}^H (P_{K,i})^{Y_i} (le_i)^{N_i}, \quad (2.6)$$

où $P_{K,i}$ est l'entrée (K, i) de T^* , le_i est la probabilité sous équilibre de liaison de l'haplotype i , et Y_i et N_i sont les nombres d'haplotypes i observés respectivement chez les cas et les témoins. Alors, $\sum_i Y_i$ est le nombre de chromosomes d'individus affectés, et $\sum_i N_i$ est le nombre de chromosomes normaux dans l'échantillon. La vraisemblance (2.6) suppose l'indépendance des individus. On sait cependant que les individus sont reliés par une généalogie complexe et inconnue; les auteurs conseillent donc d'inclure des individus qui ne sont pas trop proches, c'est-à-dire qu'ils ne doivent pas partager d'ancêtres sur quatre générations. Enfin, la vraisemblance est obtenue par la somme pondérée des H vraisemblances:

$$L = \sum_{i=1}^H f_i L_i, \quad (2.7)$$

où f_i est la fréquence de l'haplotype i dans la population, estimée sous hypothèse d'équilibre de liaison. L'haplotype ancestral le plus vraisemblable *a posteriori* est l'haplotype qui apporte le plus de poids à la vraisemblance définie par l'équation (2.7).

L'estimation des fréquences des allèles des différents marqueurs est liée à l'estimation du paramètre α , et donc, ces fréquences sont réestimées à chaque incrément de α . Par exemple, la fréquence de l'allèle x_A du marqueur A est:

$$f(x_A) = \frac{N_D f(x_A)_D (1 - \alpha) + N_N f(x_A)_N}{N_D (1 - \alpha) + N_N},$$

où N_D et N_N sont les nombres de cas et de témoins, $f(x_A)_N$ est la fréquence de l'allèle x_A pour les haplotypes normaux et $f(x_A)_D$ pour les haplotypes atteints. La vraisemblance en (2.7) est comparée à la vraisemblance sous l'hypothèse nulle, qui est générée avec l'hypothèse $\alpha = 0$. Ce ratio de vraisemblance est alors maximisé sur α , la position du locus (x), et sur g .

La méthode ne tient pas compte de la structure de la population, c'est-à-dire de la dépendance entre les chromosomes. De plus, les auteurs supposent l'équilibre de liaison, ce qui est une hypothèse très forte, surtout pour faire de la cartographie fine, car les allèles des marqueurs adjacents sont liés (sinon, il n'y a pas de déséquilibre de liaison). Service *et al.* (1999) comparent leur méthode à celle de Terwilliger (1995) et montrent qu'elle est plus puissante. Il n'y a malheureusement pas d'exemples réels qui auraient permis la comparaison avec d'autres méthodes.

2.1.5. Approche de Rannala et Slatkin

La présence d'une association non aléatoire entre allèles à deux loci, ou entre un phénotype et des allèles à un ou des marqueurs, indique une histoire récente partagée, et peut être utilisée pour estimer le taux de recombinaison ou l'âge

d'une mutation. Contrairement à d'autres méthodes, la méthode proposée par Rannala et Slatkin (1998) tient compte de plusieurs sources pour expliquer la variabilité de l'évolution et de son rôle dans l'estimation des paramètres. Sous certaines conditions, des estimés relativement précis du taux de recombinaison et du taux de mutation peuvent être obtenus.

On s'intéresse à un locus où un allèle mutant M est apparu il y a t_1 générations, et t_1 est considéré comme un paramètre. La mutation, après être apparue, est seulement présente dans une lignée de la généalogie des séquences (cf. fig. 2.1.6). Donc, t_1 est l'âge de l'allèle, alors que t_2 est l'âge de l'ancêtre commun le plus récent de l'échantillon. Ce locus est lié à un locus marqueur possédant deux allèles A_1 et A_2 . Les données sont constituées de i chromosomes porteurs de M , dont Y_0 sont porteurs de A_1 . La méthode n'est donc pas une méthode pour plusieurs loci, puisque l'on suppose la connaissance d'un seul marqueur. Le nombre Y_0 détermine la configuration des haplotypes observés. Le problème consiste à utiliser i , Y_0 et l'information sur la population pour estimer t_1 , le temps d'origine de M , c le taux de recombinaison entre M et le locus marqueur A , ou encore μ , le taux de mutation au locus marqueur.

Un estimateur couramment utilisé pour t_1 , μ ou c est l'estimateur des moments, basé sur la décroissance exponentielle du déséquilibre de liaison au cours du temps, à la suite de mutations et de recombinaisons. Cependant, la distribution de Y_0 dépend de la généalogie de M , qui elle dépend de paramètres démographiques dont les estimateurs des moments ne tiennent pas compte. Par contre, on peut considérer le cas extrême où les estimateurs des moments sont aussi des estimateurs à maximum de vraisemblance, ce qui se produit si la généalogie est en forme d'étoile.

Une mutation unique M non récurrente s'est produite il y a t_1 générations. Une description complète du processus générant la configuration des haplotypes observés présente deux composantes: (1) un modèle du processus généalogique décrivant à la fois les temps de coalescence et les relations parmi les chromosomes

porteurs de M , et (2) un modèle du processus de recombinaison entre M et le marqueur lié, ainsi qu'un modèle du processus de mutation au marqueur lié. La figure 2.1.6 illustre la généalogie considérée:

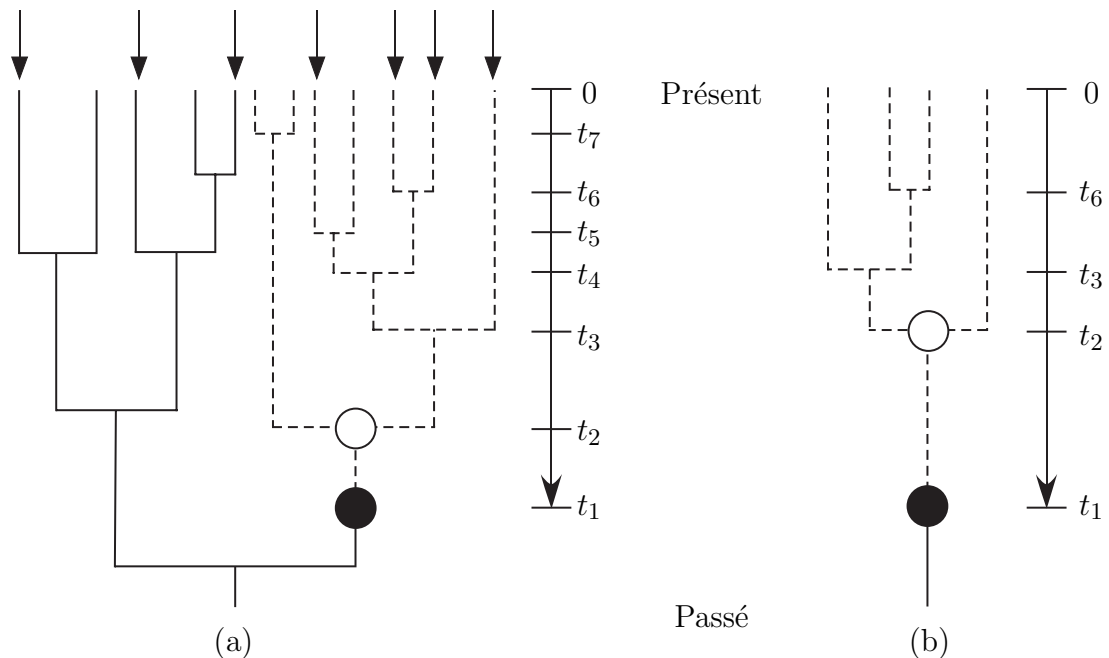


Figure 2.1.6. (a) Une mutation M (●) s'est produite au temps t_1 . Les descendances mutantes sont dénotées par les lignes hachurées. Les flèches indiquent les chromosomes échantillonnés. Le premier ancêtre commun des chromosomes porteurs de M apparaît au temps t_2 et est indiqué par un cercle vide. (b) Généalogie des quatre chromosomes porteurs de M échantillonnés de (a). Même si t_1 reste fixe, t_2 est plus récent que pour la population entière (Adapté de Rannala et Slatkin, 1998).

La distribution des temps de coalescence t_2, \dots, t_i dépend de ξ , comprenant les effets de croissance de la population et de sélection, de $f = n(0)/[2N(0)]$, la fraction de la population échantillonnée (où $N(0)$ est la taille de la population diploïde[†] au moment de l'échantillonnage, et $n(0)$ le nombre total de chromosomes échantillonnés), de i , le nombre de chromosomes porteurs de M dans l'échantillon et de t_1 , le temps d'introduction de M .

Nous allons maintenant obtenir la distribution des temps de coalescence. Soient $N(t)$ la taille de la population diploïde au temps t ($2N(t)$ chromosomes),

[†] cf. section 1.1, page 6 pour la définition de diploïde

$N(0) \gg n(0)$, $j(0)$ le nombre de chromosomes porteurs de M dans l'échantillon, $j(t)$ le nombre de chromosomes ancestraux porteurs de M au temps t ($j(t) \leq j(0)$), $n(t)$ le nombre de chromosomes ancestraux au temps t ($n(t) \geq j(t)$), et $i(t)$ le nombre de chromosomes ancestraux non porteurs de M au temps t ($n(t) = i(t) + j(t)$), où le temps est mesuré en générations.

Si $j(t)$ chromosomes ancestraux porteurs de M existent au temps t , la probabilité qu'un nouveau chromosome porteur de M apparaisse dans un intervalle de temps de longueur dt est:

$$P[j(t - dt) = j(t) + 1] = \sum_{i(t)=1}^{n(0)-j(0)} P[j(t - dt) = j(t) + 1 | i(t)] \cdot P[i(t)],$$

où

$$\begin{aligned} & P[j(t - dt) = j(t) + 1 | i(t)] \\ &= \frac{1}{4N(t)} [i(t) + j(t)] [i(t) + j(t) - 1] \left[\frac{j(t)}{i(t) + j(t)} \right] dt \\ &= \frac{1}{4N(t)} [i(t) + j(t) - 1] j(t) dt, \end{aligned}$$

en prenant en compte la probabilité d'une duplication et la probabilité qu'un chromosome dupliqué soit porteur de M . La probabilité inconditionnelle est alors:

$$P[j(t - dt) = j(t) + 1] = \sum_{i(t)=1}^{n(0)-j(0)} \frac{1}{4N(t)} [i(t) + j(t) - 1] j(t) P[i(t)] dt.$$

Si les mutants sont rares, alors $i(t) \gg j(t)$ et $i(t) \approx n(t)$, et on a:

$$\begin{aligned} P[j(t - dt) = j(t) + 1] &\approx \frac{j(t)}{4N(t)} \sum_{i(t)=1}^{n(0)-j(0)} i(t) P[i(t)] dt \\ &= \frac{j(t)}{4N(t)} E[i(t)] dt \\ &\approx \frac{j(t)}{4N(t)} E[n(t)] dt. \end{aligned}$$

Pour une population de taille variable mais déterministe, une approximation de $E[n(t)]$ est, quand cette expression est grande (Slatkin et Rannala, 1997):

$$E[n(t)] = \frac{n(0)}{1 + n(0)\tau(t)/2},$$

où $\tau(t)$ est le temps, rééchelonné pour pouvoir permettre une taille de population variable:

$$\tau(t) = \int_0^t \frac{1}{2N(t')} dt'.$$

Si la population est de taille constante, on a alors:

$$\begin{aligned} P[j(t - dt) = j(t) + 1] &= \frac{n(0)}{(1 + n(0)t/2)4N(0)} j(t) dt \\ &= \frac{f/2}{(1 + ft/2)} j(t) dt, \end{aligned}$$

où $f = n(0)/[2N(0)]$ est la fraction des chromosomes échantillonnés. Si la population a eu une croissance au taux r , on utilise une transformation de temps $\tau(t) = (e^{rt} - 1)/[2N(0)r]$ pour obtenir:

$$\begin{aligned} P[j(t - dt) = j(t) + 1] &= \frac{n(0)}{(1 + n(0)(e^{rt} - 1))[4N(0)r]} \\ &= \frac{1}{4N(t)} j(t) dt \\ &= \frac{fr}{f - (f - 2r)e^{-rt}} j(t) dt. \end{aligned}$$

C'est le taux de naissance instantané pour un processus de naissance et de mort "reconstruit".

La modélisation se fait "en avant" dans le temps, en partant de l'ancêtre pour aller vers l'échantillon. Un locus marqueur A , ayant deux allèles A_1 et A_2 , est lié à M ; la fréquence de A_1 est p parmi les chromosomes non porteurs de M , proportion supposée constante au cours du temps. Soit μ le taux de mutation de A_1 à A_2 , et ν le taux de mutation de A_2 à A_1 . Soit c le taux de recombinaison. Un modèle diploïde est considéré mais les homozygotes MM sont ignorés, car M étant rare, ils

apparaissent à une faible fréquence; les recombinaisons de chromosomes porteurs de M impliquent alors toujours des échanges avec des chromosomes non porteurs de M . Le taux de transition de MA_1 à MA_2 (par mutation ou recombinaison) est $u = [\mu + c(1 - p)]$. De façon similaire, le taux de transition de MA_2 à MA_1 est $v = [\nu + cp]$. Les probabilités de transition durant un intervalle de longueur Δt sont alors:

$$P_{\Delta t}(MA_1 \rightarrow MA_2) = \frac{u}{u+v} \left[1 - e^{-(u+v)\Delta t} \right],$$

$$P_{\Delta t}(MA_2 \rightarrow MA_1) = \frac{v}{u+v} \left[1 - e^{-(u+v)\Delta t} \right].$$

Si nous descendons la généalogie des chromosomes porteurs de M , chacun des chromosomes a autant de chance de participer à la création d'un nouveau chromosome à chaque instant. Une coalescence est alors associée à une naissance d'une nouvelle lignée mutante qui laissera des descendants dans l'échantillon (cf. fig. 2.1.7).

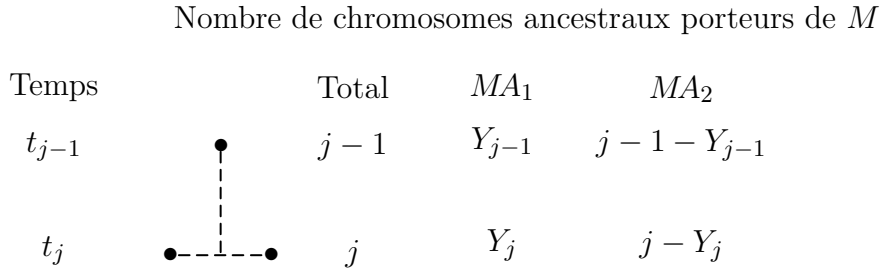


Figure 2.1.7. Schéma du nombre de chromosomes MA_1 et MA_2 lors d'un événement de coalescence.

Au temps t_j , un nouveau chromosome est formé en dupliquant l'un des $(j-1)$ chromosomes existant juste avant la duplication. Si Y_j dénote le nombre de chromosomes MA_1 au temps t_j , et si $P^*(Z|Y_{j-1})$ est la probabilité que Z chromosomes MA_1 existent immédiatement avant la duplication, sachant que Y_{j-1} existaient au temps t_{j-1} , alors:

$$P(Y_j|Y_{j-1}) = \left(\frac{Y_j - 1}{j - 1} \right) P^*(Y_j - 1|Y_{j-1}) + \left(\frac{j - 1 - Y_j}{j - 1} \right) P^*(Y_j|Y_{j-1}).$$

La probabilité que k des Y_{j-1} chromosomes MA_1 soient remplacés par des chromosomes MA_2 dans l'intervalle de temps $\Delta_t = t_{j-1} - t_j$ est:

$$P(k|Y_{j-1}) = \binom{Y_{j-1}}{k} (uG)^k (1-uG)^{Y_{j-1}-k},$$

où $G = [1 - e^{-(u+v)\Delta_t}]/(u+v)$. La probabilité que l des $j-1 - Y_{j-1}$ chromosomes MA_2 soient remplacés par des chromosomes MA_1 dans le même intervalle de temps est:

$$P(l|Y_{j-1}) = \binom{j-1-Y_{j-1}}{l} (vG)^l (1-vG)^{j-1-Y_{j-1}-l}.$$

Le nombre de chromosomes MA_1 immédiatement avant la duplication au temps t_j est alors Y'_{j-1} avec probabilité:

$$\begin{aligned} P^*(Y'_{j-1}|Y_{j-1}) &= \sum_k \binom{Y_{j-1}}{k} \times \binom{j-1-Y_{j-1}}{Y'_{j-1}-Y_{j-1}+k} \\ &\times (uG)^k (1-uG)^{Y_{j-1}-k} \times (vG)^{Y'_{j-1}-Y_{j-1}+k} (1-vG)^{j-1-Y'_{j-1}-k}. \end{aligned}$$

La probabilité de n'importe quelle configuration des haplotypes Y_0 , pour les i chromosomes porteurs de M échantillonnés, peut être calculée comme suit:

$$\begin{aligned} P(Y_0|\Theta_1) &= \sum_{Y_i=0}^i \sum_{Y_{i-1}=0}^{i-1} \dots \sum_{Y_1=0}^1 \int_{t_2=0}^{t_1} \int_{t_3=0}^{t_2} \dots \int_{t_i=0}^{t_{i-1}} P(Y_0|Y_i, \mathbf{t}; \Theta_2) \\ &\times \prod_{j=2}^i P(Y_j|Y_{j-1}, \mathbf{t}; \Theta_3) \times P(Y_1|p) \times P(\mathbf{t}|\Theta_4) dt_i \dots dt_2, \end{aligned}$$

où $\Theta_1 = \{u, v, p, t_1, i, f, \xi\}$, $\Theta_2 = \{u, v, i\}$, $\Theta_3 = \{u, v, t_1, i\}$, $\Theta_4 = \{i, f, \xi, t_1\}$ et $t = (t_2, \dots, t_i)$. Il y a $i!$ termes dans cette somme, dont chacun comprend une intégrale de dimension $i-2$. Un estimateur de Monte-Carlo de $P(Y_0|\Theta_1)$ est obtenu par:

$$P(Y_0|\Theta_1) \approx \frac{1}{R} \sum_{k=1}^R P(Y_0|\tilde{Y}_i(k), \tilde{\mathbf{t}}(k); u, v),$$

où la somme est évaluée sur les R répliqués de Monte-Carlo des variables aléatoires $\tilde{Y}_i(k)$ et $\tilde{\mathbf{t}}(k)$. Notons que $\tilde{Y}_i(k)$ est le $k^{\text{ème}}$ des R variables aléatoires indépendantes générées de la distribution:

$$P(Y_i|\mathbf{t}; \Theta_3, p) = \prod_{j=2}^i P(Y_j|Y_{j-1}, \mathbf{t}; \Theta_3)P(Y_1|p).$$

Les variables aléatoires sont générées de cette distribution par simulation séquentielle des distributions de probabilités conditionnelles, en commençant par la configuration Y_1 et en finissant par la configuration Y_i .

Les auteurs appliquent ensuite la méthode sur les données de la *dysplasie diatrophique* (DTD) publiées dans Hästbacka *et al.* (1992). DTD est une maladie autosomale récessive (seuls les individus MM sont atteints), de fréquence estimée à 0.8% en Finlande. Des études ont montré que le locus causal était à environ 70 kb (kilobase, c'est-à-dire 1000 nucléotides) du locus CSF1R. Les données sont constituées de 146 chromosomes DTD, dont 7 ont les restrictions EcoRI et StyI (haplotype 1-1) au locus CSF1R. Dans la population initiale, il n'y avait qu'une seule copie mutante de l'allèle DTD sur un chromosome portant l'haplotype 1-1 au locus marqueur. Cet haplotype a une fréquence de 3% sur des chromosomes non porteurs ($p = 0.03$). Il y a aussi dans les données un haplotype 1-2, mais il sera ignoré. En fait, les deux sites de restriction ne sont traités que comme un seul marqueur.

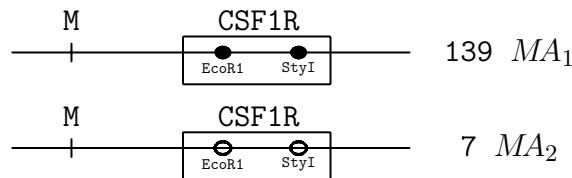


Figure 2.1.8. Schéma des données de DTD.

On estime le nombre total de chromosomes échantillonnés par $n = 146/0.008 = 18250$. La proportion de l'échantillon est $f = n(0)/[2N(0)] = 18250/[2 \times 5 \times 10^6] = 1.825 \times 10^{-3}$, en estimant à 5×10^6 la taille de la population finlandaise. On suppose que la population a été fondée par 1000 individus il y a 2000 ans, soit $t_1 = 100$ générations, et donc que la population a grandi à un taux $\xi = 0.085$ si l'unité de temps est une génération (en assumant qu'une génération est 20 ans). On suppose que le taux de mutation au locus marqueur est $\mu = 0$, donc $u = c(1 - p)$,

et $v = cp$, où c est le taux de recombinaison entre le locus causal de DTD et le marqueur.

Ces paramètres ont été utilisés et intégrés dans la méthode de Monte-Carlo pour calculer la vraisemblance des haplotypes observés en fonction du taux de recombinaison c . L'estimé obtenu est de 0.0008 (cf. fig. 2.1.9), soit environ 80 kb, ce qui est très proche de la distance déjà estimée d'environ 70 kb (Hästbacka *et al.*, 1994). Enfin, notons que compte tenu du taux de croissance élevé, l'estimé de c est insensible à t_1 .

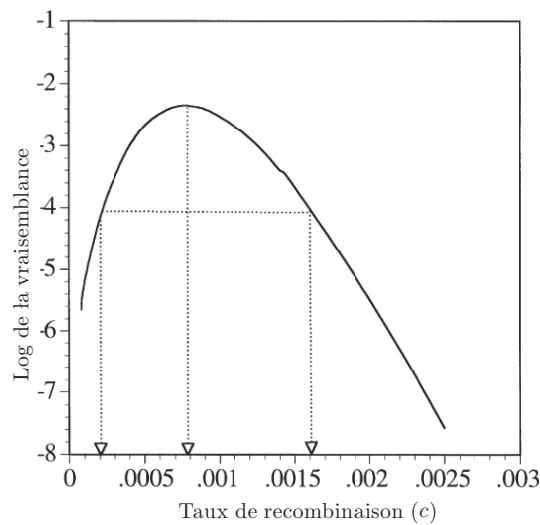


Figure 2.1.9. Vraisemblance de la position de la mutation (Rannala et Slatkin, 1998).

Voici une méthode qui modélise beaucoup mieux l'histoire des séquences. Celle-ci est décrite par un arbre, et on tient compte des événements de mutation et de recombinaisons dans l'histoire.

2.2. Le processus de coalescence

Nous avons vu que modéliser adéquatement la structure d'une population est un des grands défis des méthodes de cartographie. Or, il existe un processus simple pour décrire le passé à partir de séquences observées: le processus de coalescence. Selon Fu et Li (1999), le processus de coalescence représente le progrès

le plus significatif des vingt dernières années en génétique des populations. De nombreuses et excellentes revues ont été faites sur le sujet, parmi lesquelles celles de Hudson (1990), Donnelly et Tavaré (1995) et Nordborg (2001). Le processus de coalescence a été utilisé pour estimer des taux de mutation ou l'âge d'une allèle par exemple, mais a été peu utilisé jusqu'à présent à des fins de cartographie génétique, ce que nous proposons dans cette thèse.

Le processus de coalescence (*"The coalescent"*) est un processus stochastique à temps continu, introduit par Kingman (1982). Par coalescence, on entend l'événement par lequel deux séquences trouvent un ancêtre commun. On se place dans le cadre d'un modèle neutre de Wright-Fisher: la population diploïde de taille constante N ($2N$ chromosomes), et de très grande taille. Le modèle choisi étant neutre, les probabilités de reproduction des individus ne sont pas affectées par leur bagage génétique. Nous ne considérons pas, pour le moment, les événements de mutation et de recombinaison. Les générations se succèdent de façon discrète, et chaque nouvelle génération est formée par un échantillon aléatoire avec remise de $2N$ gamètes de la génération précédente. La distribution du nombre d'enfants

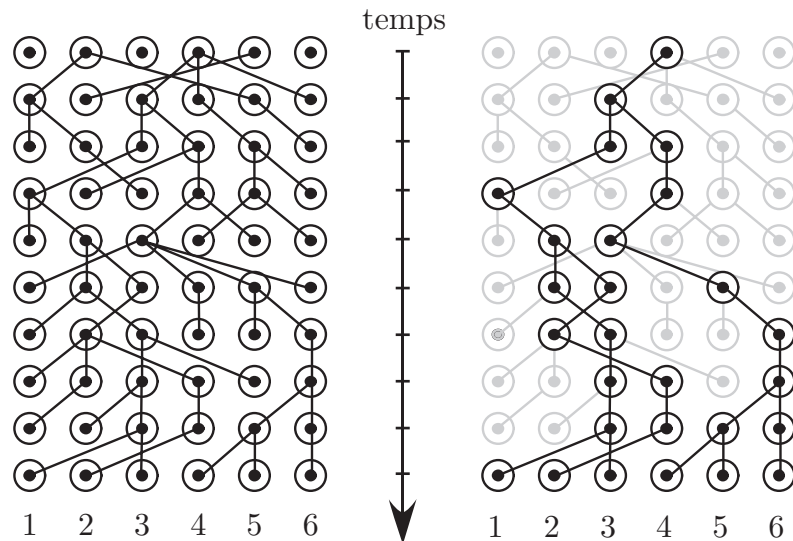


Figure 2.2.1. Réalisation d'un processus de coalescence pour $2N = 6$ sur dix générations. À gauche: toutes les relations entre les séquences sont indiquées. À droite: mise en évidence de la généalogie des six séquences de la dernière génération.

pour un parent donné est donc de loi binomiale de paramètres $2N$ et $1/(2N)$. La figure 2.2.1 (à gauche) illustre une réalisation du processus de coalescence. Comme le modèle est neutre, nous pouvons modéliser la généalogie des séquences sans tenir compte des événements de mutations qui se produisent au cours de l'évolution. Ceux-ci peuvent être ajoutés par la suite.

Si l'on s'intéresse à la généalogie des séquences de la dernière génération en remontant dans le temps (cf. fig. 2.2.1 droite), on voit que seuls quelques événements importants. Ceci a pour conséquence qu'il est très facile de simuler une telle généalogie en remontant dans le temps, puisque la majorité de la généalogie complète peut être ignorée. La généalogie des six séquences de la figure 2.2.1 peut être réorganisée pour faire apparaître une structure en arborescence (cf. fig. 2.2.2).

Soit un échantillon de n séquences, et:

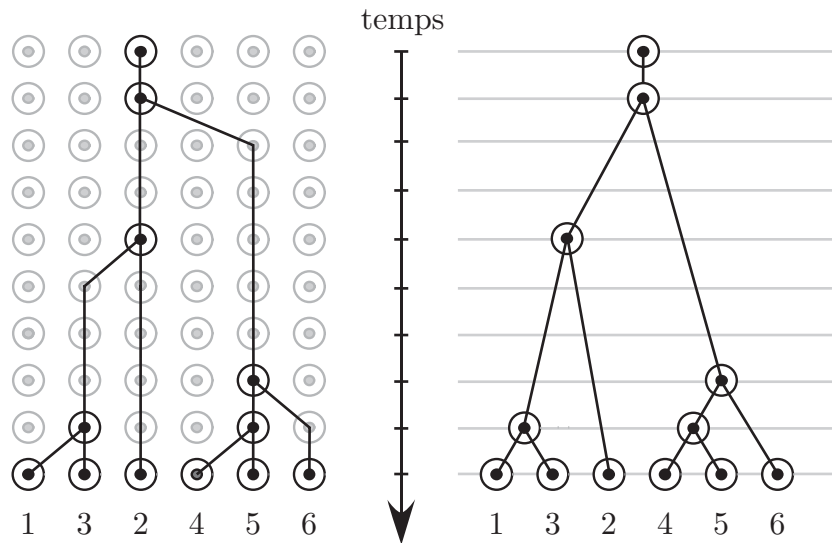


Figure 2.2.2. Réalisation d'un processus de coalescence pour $2N = 6$ (suite de la fig. 2.2.1). On ignore la partie de la généalogie ne concernant pas les séquences de la dernière génération, et on peut représenter la généalogie des six séquences par une structure en arborescence (à gauche), menant à la représentation habituelle d'un arbre de coalescence (à droite).

$P(n) = P(\text{toutes les } n \text{ séquences}$

aient des ancêtres distincts dans la génération précédente).

Comme chaque séquence provient d'un parent au hasard de la génération précédente, la probabilité que deux séquences proviennent du même parent est $1/(2N)$: la première séquence provient d'un parent au hasard parmi les $2N$, et la seconde séquence doit provenir du même parent que la première. La probabilité que deux séquences aient un ancêtre distinct est donc $1 - 1/(2N)$. La probabilité que trois séquences aient des ancêtres distincts dans la génération précédente est la probabilité que deux d'entre elles aient un ancêtre distinct ($1 - 1/(2N)$), et que la troisième ait un ancêtre distinct des deux premières, soit $(N - 2)/(2N)$. La probabilité que trois séquences aient des ancêtres distincts est donc $(1 - 1/(2N)) \times ((N - 2)/(2N))$, soit $(1 - 1/(2N)) \times (1 - 2/(2N))$. Avec les mêmes arguments, pour tout $n \geq 2$, on trouve:

$$\begin{aligned} P(n) &= \left(1 - \frac{1}{2N}\right) \left(1 - \frac{2}{2N}\right) \cdots \left(1 - \frac{n-1}{2N}\right) \\ &= 1 - \left(\frac{1+2+\cdots+n-1}{2N}\right) + O\left(\frac{1}{N^2}\right) \\ &= 1 - \frac{\binom{n}{2}}{2N} + O\left(\frac{1}{N^2}\right). \end{aligned}$$

Comme les générations se forment de façon indépendante d'une génération à la suivante, la probabilité qu'aucune coalescence ne survienne en t générations puis qu'une ait lieu à la génération $t + 1$ est donc:

$$\begin{aligned} P(n)^t [1 - P(n)] &= \left[\frac{\binom{n}{2}}{2N} + O\left(\frac{1}{N^2}\right)\right] \left[1 - \frac{\binom{n}{2}}{2N} + O\left(\frac{1}{N^2}\right)\right]^t \\ &= \lambda e^{-\lambda t} + O\left(\frac{1}{N^2}\right), \end{aligned}$$

où $\lambda = \binom{n}{2}/(2N)$. Le temps jusqu'au premier événement de coalescence est donc distribué selon une loi géométrique si le temps est mesuré en nombre de générations, qui est approximé par une loi exponentielle de moyenne $2/[n(n-1)]$

si le temps est mesuré en nombre de $2N$ générations, c'est-à-dire si $2N$ générations est l'unité de temps.

Soit W_k le temps d'attente pour que l'on ait coalescence alors que l'on a k séquences dans la généalogie (cf. fig. 2.2.3). Le temps total jusqu'à ce que l'on trouve l'ancêtre commun le plus récent (MRCA: "most recent common ancestor") de n séquences (T_{MRCA}) est donc une somme de temps d'attente:

$$T_{MRCA} = \sum_{k=2}^n W_k.$$

Le temps moyen jusqu'au MRCA est:

$$E(T_{MRCA}) = E\left[\sum_{k=2}^n W_k\right] = \sum_{k=2}^n E(W_k) = \sum_{k=2}^n \frac{2}{k(k-1)} = 2\left(1 - \frac{1}{n}\right).$$

Ainsi, on constate que l'espérance du temps moyen jusqu'à l'ancêtre commun est inférieur ou égal à 2, en unités de $2N$ générations. Aussi, on sait que $E(W_2)$, le temps moyen pour passer dans la généalogie de deux à une séquence est 1, ce qui

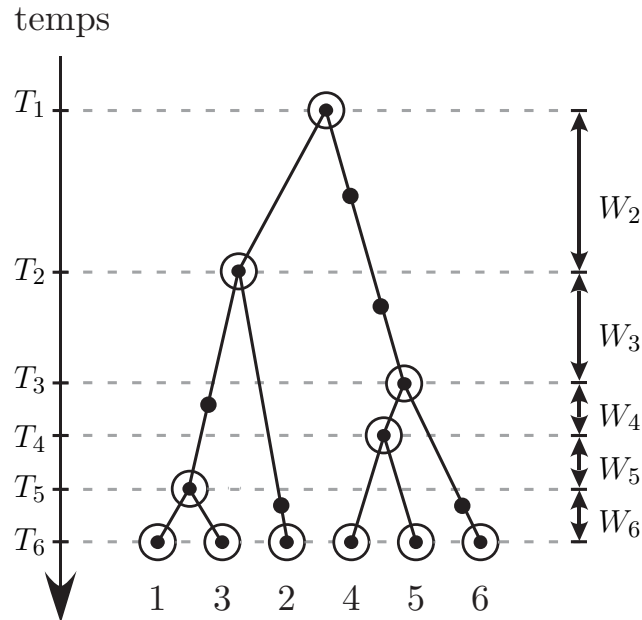


Figure 2.2.3. Arbre de coalescence, avec l'indication des temps T_i ($i = 1, \dots, 6$), et des temps d'attente entre événements de coalescence W_i ($i = 2, \dots, 6$).

signifie que le temps où seulement deux séquences sont présentes dans le graphe représente en moyenne au moins la moitié de la hauteur de la généalogie. Pour trouver la longueur totale des branches de la généalogie (T_{tot}), il suffit de pondérer le temps où l'on a un certain nombre d'ancêtres par le nombre d'ancêtres à ce temps:

$$E(T_{tot}) = E \left[\sum_{k=2}^n k \cdot W_k \right] = \sum_{k=2}^n \frac{2}{(k-1)},$$

qui est asymptotique à $2(\gamma + \log(n))$, où $\gamma = 0.57721566$ est la constante d'Euler.

Soit u la probabilité d'un événement de mutation par séquence par génération. Un événement de mutation ne change pas le nombre de séquences, il modifie seulement une séquence en changeant un allèle par un autre. En représentant un événement de coalescence par Co et un événement de mutation par Mu , et en définissant $\theta = 4Nu$, on obtient:

$$P(Co | Co \text{ ou } Mu) = \frac{\binom{n}{2}/(2N) + O\left(\frac{1}{N^2}\right)}{\binom{n}{2}/(2N) + nu + O\left(\frac{1}{N^2}\right)} = \frac{n-1}{n-1+\theta} + O\left(\frac{1}{N}\right),$$

$$P(Mu | Co \text{ ou } Mu) = \frac{nu}{\binom{n}{2}/(2N) + 2u + O\left(\frac{1}{N^2}\right)} = \frac{\theta}{\theta + n - 1} + O\left(\frac{1}{N}\right).$$

Le nombre de générations avant qu'un événement de coalescence ou de mutation ne se produise est distribué géométriquement avec paramètre:

$$\begin{aligned} & 1 - \left(1 - \frac{\binom{n}{2}}{2N} + O\left(\frac{1}{N^2}\right) \right) (1-u)^n \\ &= \frac{n(n-1) + n\theta}{4N} + O\left(\frac{1}{N^2}\right), \end{aligned}$$

en notant que u est une fonction $O\left(\frac{1}{N}\right)$ pour θ fixé. En prenant $2N$ générations comme unité de temps, et en faisant tendre N vers l'infini, le taux avec lequel un événement se produit est :

$$\frac{n(n-1) + n\theta}{2},$$

le taux de mutation:

$$\nu = \frac{n(n-1) + n\theta}{2} \times \frac{\theta}{\theta + n - 1} = \frac{n\theta}{2},$$

et le taux de mort par coalescence:

$$\mu = \frac{n(n-1) + n\theta}{2} \times \frac{n-1}{\theta + n - 1} = \frac{n(n-1)}{2}.$$

Le processus de coalescence est un outil très flexible. Ainsi, des développements ont été faits pour tenir compte des structures de population, des tailles de population variable et de la sélection. On peut consulter Nordborg (2001) pour une synthèse de tous ces développements. Ainsi, nous avons là une façon simple de construire des généalogies de séquences, en partant des séquences observées jusqu'à l'ancêtre commun. Cependant, pour réaliser nos objectifs de cartographie, il est important de tenir compte des événements de recombinaison pouvant affecter l'histoire des séquences. C'est ce que nous allons voir dans la prochaine section.

2.3. Le graphe de recombinaison ancestral

Le graphe de recombinaison ancestral (ou "ARG" pour "Ancestral Recombination graph") est une extension du processus de coalescence proposée par Hudson (1983, 1990) et développée par Griffiths et Marjoram (1996, 1997). Il s'agit de modéliser l'évolution de séquences en remontant dans le temps, comme dans le processus de coalescence, mais en intégrant à la généalogie des gènes un nouvel événement: la recombinaison (*Re*). Quand un événement de recombinaison se produit, la séquence est séparée en deux au point de recombinaison (nous supposons ici que la séquence est petite, donc que plusieurs événements de recombinaison sont impossibles dans la même méiose), tel qu'illustré dans la figure 2.3.1

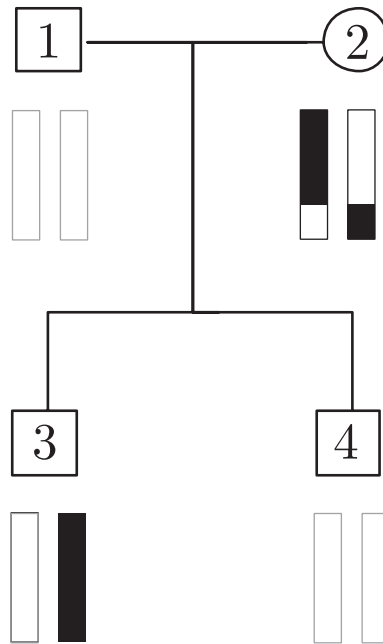


Figure 2.3.1. Illustration de la transmission du matériel génétique d'une séquence (fragment du chromosome maternel de l'individu 3, en noir) lors d'un événement de recombinaison dans le contexte de l'ARG.

La figure 2.3.2 présente l'exemple d'une généalogie familiale sur neuf générations, où l'on suit l'histoire du matériel génétique de quatre séquences de la dernière génération jusqu'à leur ancêtre commun. Deux événements de mutation et deux événements de recombinaison se produisent dans l'histoire des quatre séquences échantillonnées. La figure 2.3.3 présente le graphe de recombinaison ancestral correspondant à l'échantillon des quatre séquences de la généalogie de la figure 2.3.2.

Bien que le linkage et le déséquilibre de liaison aient longtemps été vus comme deux approches différentes de faire de la cartographie, il est facile de voir avec le graphe de recombinaison ancestral que ces deux approches ne sont pas si éloignées l'une de l'autre. Comme le fait remarquer Nordborg (2001), la différence majeure tient à la façon d'échantillonner. En linkage, comme on l'a vu précédemment, on échantillonne les membres d'une même famille et la généalogie est habituellement connue, alors que pour le déséquilibre de liaison,

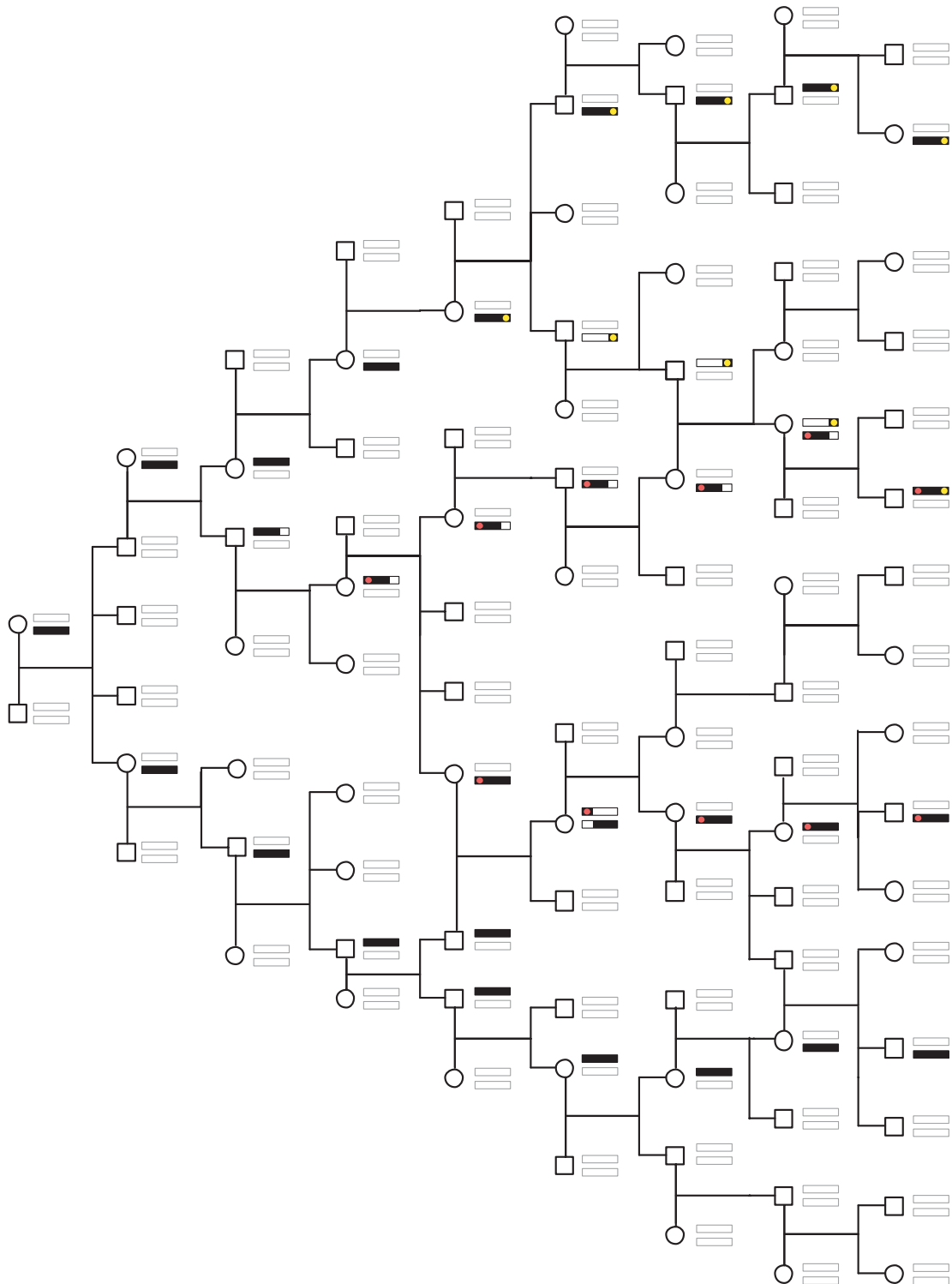


Figure 2.3.2. Généalogie familiale sur neuf générations, où le matériel génétique de quatre séquences de la dernière génération est suivi jusqu'à leur ancêtre commun. Deux recombinaisons et deux mutations (● et ●) se produisent dans ce matériel.

on échantillonne des séquences au hasard dans la population et la généalogie sous-jacente des séquences échantillonnées est inconnue. Mais il est clair que les séquences prises d'une population sont également apparentées (nous sommes tous des humains), mais à un degré moindre.

La généalogie de la figure 2.3.2 est petite, mais on peut la considérer comme une minuscule population. Évidemment, dans une application réelle, un nombre beaucoup plus grand de séquences seraient échantillonnées pour mener une étude utilisant le déséquilibre de liaison, et la généalogie sous-jacente serait beaucoup plus longue. Comme on peut le voir dans la figure 2.3.3, l'ARG ne retient que les événements concernant l'histoire des séquences observées: c'est-à-dire que tous les autres événements de la généalogie n'ayant aucune influence sur l'histoire des séquences observées sont écartés.

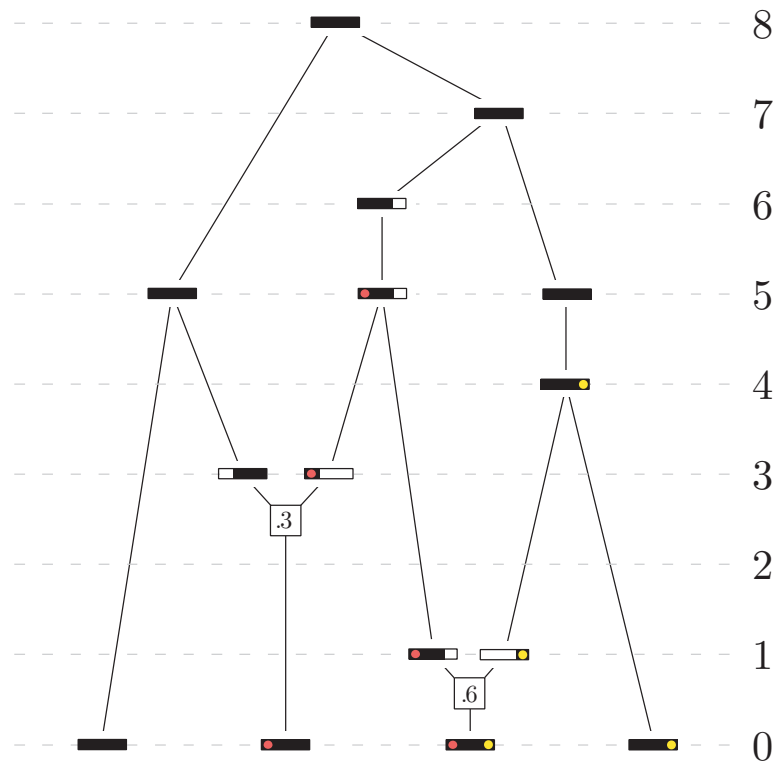


Figure 2.3.3. Graphe de recombinaison ancestral résultant de la généalogie de la figure 2.3.2 pour l'échantillon de quatre séquences. Les séquences sont représentées par l'intervalle $[0, 1]$. Deux recombinaisons se produisent aux points 0.3 et 0.6. Les mutations sont ● et ●.

Soit r la probabilité de recombinaison par séquence par génération, et $\rho = 4Nr$, fixé de telle sorte que r est une fonction $O(1/N)$. Soit n le nombre de séquences. Alors, en une génération vers le passé, on a :

$$\begin{aligned} P(Co) &= \frac{n(n-1)}{4N} + O\left(\frac{1}{N^2}\right), \\ P(Re \text{ pour la séquence } i) &= r, \\ P(Mu \text{ pour la séquence } i) &= u. \end{aligned}$$

Le temps avant qu'un quelconque de ces événements ne se produise est distribué géométriquement avec pour paramètre :

$$\begin{aligned} &1 - \left(1 - \frac{n(n-1)}{4N} + O\left(\frac{1}{N^2}\right)\right) (1-r)^n (1-u)^n \\ &= 1 - \left(1 - \frac{n(n-1)}{4N}\right) (1-nr)(1-nu) + O\left(\frac{1}{N^2}\right) \\ &= \frac{n(n-1) + n\rho + n\theta}{4N} + O\left(\frac{1}{N^2}\right). \end{aligned}$$

La probabilité qu'il y ait une coalescence, une mutation ou une recombinaison, étant donné que l'un de ces événements se produit est :

$$\begin{aligned} P(Co | Co \text{ ou } Re \text{ ou } Mu) &= \frac{\binom{n}{2}/(2N) + O\left(\frac{1}{N^2}\right)}{\binom{n}{2}/(2N) + O\left(\frac{1}{N^2}\right) + nu + nr} \\ &= \frac{n-1}{n-1 + \theta + \rho} + O\left(\frac{1}{N}\right), \end{aligned}$$

$$\begin{aligned} P(Mu | Co \text{ ou } Re \text{ ou } Mu) &= \frac{nu}{\binom{n}{2}/(2N) + nu + nr + O\left(\frac{1}{N^2}\right)} \\ &= \frac{\theta}{n-1 + \theta + \rho} + O\left(\frac{1}{N}\right), \end{aligned}$$

$$\begin{aligned} P(Re | Co \text{ ou } Re \text{ ou } Mu) &= \frac{nr}{\binom{n}{2}/(2N) + nr + nu + O\left(\frac{1}{N^2}\right)} \\ &= \frac{\rho}{n-1 + \theta + \rho} + O\left(\frac{1}{N}\right). \end{aligned}$$

En prenant $2N$ générations comme unité de temps et en faisant tendre N vers l'infini, le taux de changement de n séquences est $(n(n-1) + n\rho + n\theta)/2$, et les

taux de naissance (recombinaison) et de mort (coalescence) du processus sont respectivement:

$$\nu = \frac{n(n-1) + n\theta + n\rho}{2} \times \frac{\rho}{n-1 + \theta + \rho} = \frac{n\rho}{2},$$

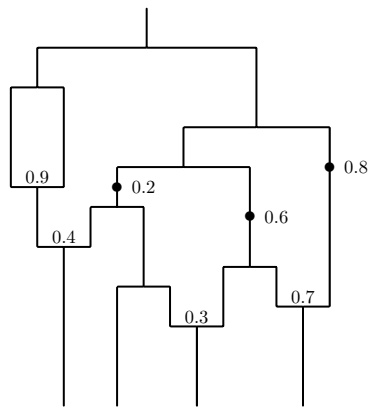
$$\mu = \frac{n(n-1) + n\theta + n\rho}{2N} \times \frac{n-1}{n-1 + \theta + \rho} = \frac{n(n-1)}{2}.$$

Le taux de mort étant de l'ordre de n^2 alors que le taux de naissance est seulement linéaire en n , il y a un ancêtre commun dans le graphe en un temps fini avec probabilité 1. Il est implicite que le processus est défini en arrière dans le temps jusqu'à l'infini négatif, mais le graphe présente habituellement un intérêt seulement jusqu'au premier ancêtre commun des séquences, c'est-à-dire quand le nombre des lignées ancestrales est 1. L'inclusion de la recombinaison dans le processus de coalescence crée des dépendances. Imaginons une séquence de deux marqueurs; si la distance ρ entre eux est grande, les arbres de coalescence pour chacun de ces marqueurs sont indépendants. Par contre, si ρ est petit, ils sont dépendants.

2.4. Fondements théoriques de l'ARG

Nous allons maintenant présenter les fondements théoriques de l'ARG par l'étude d'une série de théorèmes énoncés dans Griffiths et Marjoram (1997). Ils sont importants puisqu'ils stipulent, entre autres choses, que de multiples événements de recombinaison sont rares et que l'espérance du temps jusqu'à l'ancêtre commun est bornée.

Quelques détails contextuels diffèrent de ce que l'on a supposé jusqu'ici dans ce travail. Une séquence est ici représentée par l'intervalle $[0, 1]$, et le temps est mesuré en unités de $2N$ générations, $2N$ tendant vers l'infini et représentant le nombre total de séquences dans la population. La probabilité d'une recombinaison par séquence par génération est r , et la probabilité d'une mutation par séquence par génération est u . On définit $\rho = 4Nr$, et $\theta = 4Nu$. La figure 2.4.1 (a) montre un exemple de graphe de recombinaison ancestral dans sa représentation usuelle.



(a) Exemple d'ARG.

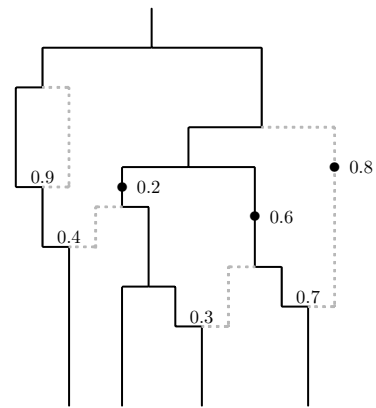
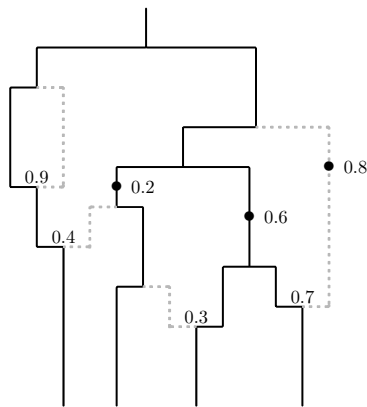
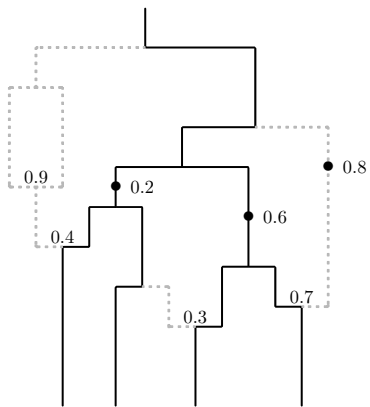
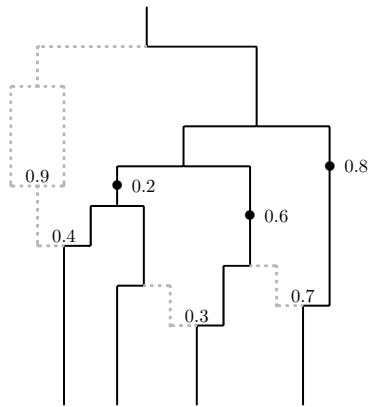
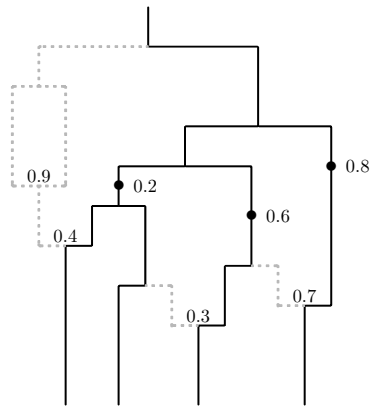
(b) Graphe partiel pour $x \in [0, 0.3]$ (c) Graphe partiel pour $x \in (0.3, 0.4]$ (d) Graphe partiel pour $x \in (0.4, 0.7]$ (e) Graphe partiel pour $x \in (0.7, 0.9]$ (f) Graphe partiel pour $x \in (0.9, 1]$

Figure 2.4.1. (a) Exemple de graphe de recombinaison ancestral. Les mutations sont représentées par des \bullet , avec l'indication de leur coordonnée. Une recombinaison se produit quand une arête se sépare en deux, alors qu'une coalescence se produit quand deux arêtes se confondent dans une. La coordonnée du point de recombinaison est également indiquée. (b)–(f) arbres de coalescence marginaux du graphe (a). Les arêtes représentées en pointillé gris n'appartiennent pas aux arbres marginaux.

Soit k le nombre de séquences dans le graphe à un instant donné, et n le nombre de séquences échantillonnées. Comme nous l'avons vu, le nombre d'ancêtres $\{\xi_t, t \geq 0\}$ est un processus de naissance et de mort, dont les taux respectifs sont $\mu_k = k(k-1)/2$ et $\lambda_k = k\rho/2$. Chaque point $x \in [0, 1]$ a un arbre de coalescence marginal $T(x)$. On a un nombre fini d'arbres marginaux: il y en a deux par point de recombinaison, donc on a au maximum 2^R arbres marginaux si on a R événements de recombinaisons (certains des 2^R arbres peuvent être identiques). La figure 2.4.1 (b)–(f) illustre tous les arbres de coalescence marginaux de la figure 2.4.1 (a). Pour construire un arbre marginal au point $x \in [0, 1]$, on suit les arêtes du graphe de recombinaison ancestral; quand une recombinaison est rencontrée, on prend le chemin de gauche si x est inférieur ou égal au point de recombinaison, sinon on prend le chemin de droite. L'ensemble de ces arêtes définit l'arbre marginal $T(x)$ au point x . Le grand MRCA est l'ancêtre de toutes les séquences, c'est-à-dire le point le plus haut dans le graphe tel que le nombre de séquences est un. Par ailleurs, pour un point $x \in [0, 1]$ donné, il existe un MRCA local: il correspond au point le plus haut dans $T(x)$. Ainsi, si l'on prend l'exemple de l'ARG de la figure 2.4.1 (a), on peut voir que pour $x \in [0, 0.03]$ le MRCA local est aussi le grand MRCA, que le MRCA local est le même pour $x \in (0.7, 0.9]$ et $x \in (0.9, 1]$. Enfin, un point x dans $[0, 1]$ est ancestral s'il appartient à $T(x)$. Par exemple, le matériel génétique représenté en noir dans la figure 2.3.3 de la section précédente est ancestral, s'il est représenté en blanc, il ne l'est pas.

Théorème 2.1. *Soit $R_{n,x,\delta}$ le nombre d'événements de recombinaison dans $[x - \delta/2, x + \delta/2]$, pour n séquences, avant le MRCA au point x . Soit $h_n(x)$ la densité de recombinaison à x dans $[0, 1]$, c'est-à-dire:*

$$h_n(x) = \lim_{\delta \rightarrow 0} \frac{1}{\delta} P(R_{n,x,\delta} = 1).$$

Alors on a:

$$h_n(x) = \sum_{k=1}^{n-1} \frac{\rho}{k}$$

et

$$\lim_{\delta \rightarrow 0} \frac{1}{\delta} P(R_{n,x,\delta} > 1) = 0.$$

Preuve. Si on a k ancêtres du fragment $[x - \delta/2, x + \delta/2]$, on a coalescence au taux $k(k-1)/2$ et recombinaison au taux $k\rho\delta/2$. Donc on a:

$$\begin{aligned} & P(\text{Co avant Re dans } [x - \delta/2, x + \delta/2] \mid k \text{ ancêtres}) \\ &= \frac{k(k-1)/2}{k(k-1)/2 + k\rho\delta/2} = \frac{k-1}{k-1 + \rho\delta}, \end{aligned}$$

d'où

$$\begin{aligned} & P(\text{aucune recombinaison dans } [x - \delta/2, x + \delta/2] \text{ avant le MRCA à } x) \\ &= \prod_{k=2}^n \frac{k-1}{k-1 + \rho\delta}. \end{aligned}$$

Alors on a:

$$\lim_{\delta \rightarrow 0} \frac{1}{\delta} P(R_{n,x,\delta} \geq 1) = \lim_{\delta \rightarrow 0} \frac{1}{\delta} \left(1 - \prod_{k=2}^n \frac{k-1}{k-1 + \rho\delta} \right).$$

On applique la règle de l'Hôpital:

$$\begin{aligned} \frac{\partial}{\partial \delta} \left(1 - \prod_{k=2}^n \frac{k-1}{k-1 + \rho\delta} \right) &= \frac{\rho}{(1 + \rho\delta)^2} \left(\prod_{k=3}^n \frac{k-1}{k-1 + \rho\delta} \right) \\ &+ \left(\prod_{k=2}^2 \frac{k-1}{k-1 + \rho\delta} \right) \frac{2\rho}{(2 + \rho\delta)^2} \left(\prod_{k=4}^n \frac{k-1}{k-1 + \rho\delta} \right) \\ &+ \left(\prod_{k=2}^3 \frac{k-1}{k-1 + \rho\delta} \right) \frac{3\rho}{(3 + \rho\delta)^2} \left(\prod_{k=5}^n \frac{k-1}{k-1 + \rho\delta} \right) + \dots \\ &+ \left(\prod_{k=2}^{n-1} \frac{k-1}{k-1 + \rho\delta} \right) \frac{(n-1)\rho}{(n-1 + \rho\delta)^2}. \end{aligned}$$

Donc, on obtient:

$$\lim_{\delta \rightarrow 0} \frac{1}{\delta} P(R_{n,x,\delta} \geq 1) = \rho + \frac{\rho}{2} + \frac{\rho}{3} + \dots + \frac{\rho}{n-1} = \sum_{k=1}^{n-1} \frac{\rho}{k}.$$

De plus, quel que soit le nombre d'ancêtres, le taux de recombinaison est proportionnel à δ , d'où

$$P(R_{n,x,\delta} > 1) = o(\delta),$$

ce qui complète la démonstration.

Corollaire 2.1. *Le nombre moyen d'événements de recombinaison avant de rejoindre les MRCAs marginaux le long des séquences à partir de n séquences est:*

$$\int_0^1 h_n(x) dx = \sum_{k=1}^{n-1} \frac{\rho}{k}.$$

Ceci a aussi été démontré par Hudson et Kaplan (1985).

Tel que l'ARG est défini, une recombinaison peut se produire sur n'importe quelle séquence, qu'elle soit complètement ancestrale ou non. Donc, des événements de recombinaison peuvent se produire dans des parties de séquences non ancestrales, et alors ils ne modifient pas l'histoire des séquences.

Théorème 2.2. *Soit R_n le nombre d'événements de recombinaison affectant la généalogie de n séquences, R_n^0 le nombre d'événements de recombinaison arrivant aux ancêtres n'ayant pas eu d'événements de recombinaison auparavant, et R_n^1 le nombre total d'événements de recombinaison pour aller au grand MRCA. Alors on a:*

$$R_n^0 \leq R_n \leq R_n^1, \quad (2.8)$$

$$\sum_{j=2}^n \frac{\rho}{\rho + j - 1} \leq E(R_n^0) \leq 1 + \sum_{j=2}^n \frac{\rho}{\rho + j - 1}, \quad (2.9)$$

$$E(R_n^1) = \rho \int_0^1 \frac{1 - (1-x)^{n-1}}{x} e^{\rho x} dx.$$

Quand n tend vers l'infini, $E(R_n^0)$, $E(R_n)$ et $E(R_n^1)$ sont asymptotiques à $\rho \log n$, et $E(R_n^1 - R_n^0)$ est uniformément borné pour $n \geq 2$.

Preuve. Compte tenu des définitions de R_n , R_n^0 et R_n^1 , (2.8) est évident: R_n^1 , le nombre total d'événements de recombinaison est plus grand que R_n , car R_n ne tient pas compte des événements de recombinaison n'affectant pas le matériel ancestral. Ensuite, ne considérant pas les ancêtres ayant déjà eu des événements de recombinaison, le nombre des événements les affectant R_n^0 est plus petit que le nombre d'événements de recombinaison affectant toutes les séquences.

Pour démontrer (2.9), il faut considérer un graphe où l'on supprimerait toutes les séquences ayant déjà eu un événement de recombinaison, considérer le taux

avec lequel ces lignées sont supprimées, et enfin considérer le dernier événement de recombinaison. On peut consulter Griffiths et Marjoram (1997) pour les détails de la preuve. Les autres résultats ne proviennent pas de cet article, et sont démontrés dans Ethier et Griffiths (1990).

Théorème 2.3. *Soit $p_n(x)$ la probabilité que le MRCA de $T(x-)$ et celui de $T(x+)$ soient identiques dans le graphe, sachant qu'un événement de recombinaison a eu lieu au point x avant le MRCA de $T(x)$. Alors on a :*

$$p_n(x) = 1 - \frac{n^2 + n - 2}{n(n+1) \sum_{k=1}^{n-1} \frac{1}{k}},$$

qui est asymptotique à $1 - \frac{1}{\log(n)}$ quand n tend vers l'infini.

Preuve. Il est suffisant de considérer seulement l'événement de recombinaison en question, puisque les autres n'affectent pas $p_n(x)$. Supposons que l'on ait l ancêtres quand cet événement se produit. Les deux MRCAs à droite et à gauche de x sont distincts si et seulement si une des deux dernières lignées dans la coalescence des $l+1$ gènes après l'événement de recombinaison est une lignée ancestrale pure de l'un des gènes impliqués, et non une lignée ancestrale d'aucun autre des l gènes. Ceci arrive seulement si le sous-arbre des l gènes coalesce d'abord. Il faut donc trouver la probabilité de cette configuration pour obtenir le résultat. Les détails de la preuve peuvent être trouvés dans Griffiths et Marjoram (1997).

Théorème 2.4. *Soit α_n le nombre attendu de MRCAs distincts. Alors on a :*

$$\alpha_n \leq 1 + \left(1 - \frac{2}{n^2 + n}\right) \rho,$$

qui est borné par $1 + \rho$ pour $n \geq 2$.

Preuve. On a:

$$\begin{aligned}
\alpha_n &\leq 1 + E(\text{nombre de changements d'ancêtre le long de la séquence}) \\
&= 1 + \int_0^1 P(\text{changement d'ancêtre à } x \mid \text{Re à } x) h_n(x) dx \\
&= 1 + \int_0^1 (1 - p_n(x)) h_n(x) dx \\
&= 1 + \int_0^1 \frac{n^2 + n - 2}{n(n+1)} \sum_{k=1}^{n-1} \frac{1}{k} \rho dx \\
&= 1 + \rho \int_0^1 \frac{n^2 + n - 2}{n(n+1)} dx \\
&= 1 + \rho \left(1 - \frac{2}{n^2 + n} \right).
\end{aligned}$$

Soit $\{W_n(x), 0 \leq x \leq 1\}$ l'ensemble des temps d'attente jusqu'aux MRCAs aux positions x . Marginalement, $W_n(x) = T_n(x) + \dots + T_2(x)$, où $\{T_k(x), k = 2, \dots, n\}$ sont les temps alors que l'on a k ancêtres, mutuellement indépendants. La distribution marginale $W_n(x)$ est indépendante de x et de l'histoire de recombinaison, mais la distribution $\{W_n(x), 0 \leq x \leq 1\}$ en dépend.

Théorème 2.5. Si $W = \max_{0 \leq x \leq 1} W_n(x)$, alors on a:

$$E(W) \leq 2 \left(1 - \frac{1}{n} \right) + \rho \frac{n^2 + n - 2}{2n(n+1)},$$

qui est borné par $2 + \frac{\rho}{2}$ pour $n \geq 2$.

Preuve. Soit R_n le nombre d'événements de recombinaison avant les MRCAs des n séquences, se produisant au point x_1, \dots, x_{R_n} . Prenons $x_0 = 0$ et $x_{R_n+1} = 1$. Soit W_i le temps au MRCA pour l'intervalle $(x_i, x_{i+1}]$. Alors on a:

$$\begin{aligned}
W &= \max_{i=0, \dots, R_n} W_i \\
&\leq W_0 + \sum_{i=1}^{R_n} (W_i - W_{i-1})^+,
\end{aligned}$$

où $(W_i - W_{i-1})^+$ prend la valeur $W_i - W_{i-1}$ si celle-ci est positive et 0 sinon. On a donc:

$$E(W) \leq 2 \left(1 - \frac{1}{n} \right) + E(R_n) E((W_1 - W_0)^+),$$

car $E(W_0) = 2(1 - \frac{1}{n})$, W_0 étant le temps d'attente marginal au MRCA à $x_0 = 0$, et les variables $W_i - W_{i-1}$ sont identiquement distribuées, indépendantes entre elles, et indépendantes de R_n . Maintenant, $P(W_1 > W_0) = \frac{1}{2}(1 - p_n(x_1))$ et $E(W_1 - W_0 | W_1 > W_0) = 1$ puisque $W_1 - W_0$ est le temps pris pour les deux dernières lignées de la généalogie à x_1 pour coalescer sachant que $W_1 > W_0$. Donc on a :

$$\begin{aligned} E(W) &\leq 2 \left(1 - \frac{1}{n}\right) + \frac{1}{2} E(R_n)(1 - p_n(x_1)) \\ &= 2 \left(1 - \frac{1}{n}\right) + \rho \frac{n^2 + n - 2}{2n(n+1)}, \end{aligned}$$

car $E(R_n) = \rho \sum_{k=1}^{n-1} (1/k)$ (cf. corollaire 2.1)

Maintenant que nous avons vu que nous possédions un moyen de modéliser des généalogies qui possèdent des propriétés intéressantes, comme celles que les temps d'attente entre événements sont bornés et que le nombre d'ancêtres distincts est fini, il nous faut étudier les moyens de produire des estimations dans l'ARG, ce que nous allons faire dans la prochaine section.

2.5. Méthodes d'estimation

2.5.1. Méthode de Griffiths et Tavaré

La méthode d'estimation utilisée dans ce travail, connue sous le nom de "méthode de Griffiths et Tavaré" est apparue dans la littérature en 1994 (Griffiths et Tavaré, 1994c). L'intérêt était de développer une méthode pour estimer le taux de mutation (θ) sur une séquence d'ADN par maximum de vraisemblance en utilisant le processus de coalescence. La recombinaison n'est pas considérée ici, mais comme nous le verrons plus tard dans ce travail, la méthode de Griffiths et Tavaré peut être adaptée facilement pour tenir compte des recombinaisons.

On se place ici dans le cadre du processus de coalescence qui peut être décrit par un arbre. Le nombre d'ancêtres dans l'arbre est un processus de mort. Pour n séquences, le taux de mutation est $n\theta/2$ et le taux de coalescence de $n(n-1)/2$.

Soit $Q(\mathbf{n})$ la probabilité que l'échantillon de n séquences ait la configuration $\mathbf{n} = (n_1, \dots, n_d)$, où n_i est le nombre de séquences de type i ($i = 1, \dots, d$). Le premier événement en arrière dans le temps est une mutation avec probabilité $\theta/(n + \theta - 1)$, et une coalescence avec probabilité $(n - 1)/(n + \theta - 1)$. Quand une mutation se produit, on a transition du type i au type j en avant dans le temps selon l'entrée p_{ij} de la matrice de transition P , matrice régulière avec distribution stationnaire π . Si le dernier événement est un événement de mutation, alors la configuration des séquences un pas en arrière dans le temps est \mathbf{n} , et une transition de i vers i (pour un certain i) s'est produite (donc pas de changement observable), ou $\mathbf{n} - \mathbf{e}_i + \mathbf{e}_j$, et une transition de i vers j s'est produite pour certains i et j ($i \neq j$, \mathbf{e}_i est le vecteur avec 1 à la position i et 0 ailleurs). Si le dernier événement est un événement de coalescence, alors la configuration des séquences un pas en arrière dans le temps est $\mathbf{n} - \mathbf{e}_j$ pour un certain j . Ceci conduit à l'équation de récurrence:

$$Q(\mathbf{n}) = \frac{\theta}{n + \theta - 1} \left(\sum_i \frac{n_i}{n} p_{ii} Q(\mathbf{n}) + \sum_{\substack{i,j, \\ n_j > 0, i \neq j}} \frac{n_i + 1}{n} p_{ij} Q(\mathbf{n} + \mathbf{e}_i - \mathbf{e}_j) \right) \\ + \frac{n - 1}{n + \theta - 1} \sum_{j, n_j > 0} \frac{n_j - 1}{n - 1} Q(\mathbf{n} - \mathbf{e}_j)$$

En posant $b(\mathbf{n}) = 1 - \frac{\theta}{n + \theta - 1} \sum_i \frac{n_i}{n} p_{ii}$, et $a(\mathbf{n}) = [\theta/(n(n + \theta - 1))] \sum_{i,j, n_j > 0, i \neq j} (n_i + 1) p_{ij} + 1/(n + \theta - 1) \sum_{j, n_j > 0} (n_j - 1)$, l'équation de récurrence peut s'écrire:

$$Q(\mathbf{n}) = \frac{a(\mathbf{n})}{b(\mathbf{n})} \left[\sum_{\substack{i,j, \\ n_j > 0, i \neq j}} \frac{\theta(n_i + 1)}{n(n + \theta - 1)a(\mathbf{n})} p_{ij} Q(\mathbf{n} + \mathbf{e}_i - \mathbf{e}_j) \right. \\ \left. + \sum_{j, n_j > 0} \frac{n_j - 1}{(n + \theta - 1)a(\mathbf{n})} Q(\mathbf{n} - \mathbf{e}_j) \right] \\ = f(\mathbf{n}) \left[\sum_{\substack{i,j, \\ n_j > 0, i \neq j}} \lambda_{ij}(\mathbf{n}) Q(\mathbf{n} + \mathbf{e}_i - \mathbf{e}_j) + \sum_{j, n_j > 0} \mu_j(\mathbf{n}) Q(\mathbf{n} - \mathbf{e}_j) \right],$$

où $f(\mathbf{n}) = a(\mathbf{n})/b(\mathbf{n})$, $\lambda_{ij}(\mathbf{n}) = \frac{\theta(n_i+1)}{n(n+\theta-1)a(\mathbf{n})}p_{ij}$ et $\mu_j(\mathbf{n}) = \frac{n_j-1}{(n+\theta-1)a(\mathbf{n})}$. Il faut aussi que $Q(\mathbf{e}_i) = \pi_i^*$ ($i = 1, \dots, d$), où π_i^* est la probabilité que l'ancêtre commun le plus récent soit de type i . Il est courant de supposer que $\pi_i^* = \pi_i$, où $\pi = (\pi_1, \dots, \pi_d)$ est la distribution stationnaire de P .

Pour exploiter l'équation de récurrence précédente, considérons la chaîne de Markov $\{N_k; k \geq 0\}$, dont l'espace des états est $S = \{0, 1, \dots\}^d$ et les transitions sont:

$$\begin{aligned} \mathbf{n} &\rightarrow \mathbf{n} + \mathbf{e}_i - \mathbf{e}_j && \text{avec probabilité } \lambda_{ij}(\mathbf{n}), \\ \mathbf{n} &\rightarrow \mathbf{n} - \mathbf{e}_j && \text{avec probabilité } \mu_j(\mathbf{n}). \end{aligned}$$

Notons que par construction, pour tout \mathbf{n} dans S :

$$\sum_{\substack{i,j \\ n_j > 0, i \neq j}} \lambda_{ij}(\mathbf{n}) + \sum_{j, n_j > 0} \mu_j(\mathbf{n}) = 1.$$

Soit $\{N_0, \dots, N_\tau\}$ le chemin du processus commençant à $N_0 = \mathbf{n}$ jusqu'à absorption dans l'ensemble $A = \{e_i\}$, $i = 1, \dots, d$, à τ . Alors:

$$Q(\mathbf{n}) = E_{\mathbf{n}} \prod_{k=0}^{\tau} f(N_k)$$

où $f(N_k) = a(N_k)/b(N_k)$. Ceci est une conséquence du lemme suivant:

Lemme. Soit $\{X_k; k \geq 0\}$ une chaîne de Markov d'espace d'états S et de matrice de transition $P = (p_{xy})_{x,y \in S}$. Soit A un ensemble d'états pour lequel le temps d'absorption donné par:

$$\eta \equiv \eta_A = \inf\{k \geq 0 : x_k \in A\}$$

est fini avec probabilité 1 en démarrant d'un état x dans $T \equiv S \setminus A$. Pour $f \geq 0$ une fonction sur S , définissons:

$$u_x(f) = E \left(\prod_{k=0}^{\eta} f(X_k) \mid X_0 = x \right)$$

pour tout x dans S , de telle sorte que $u_x(f) = f(x)$ pour tout x dans A . Alors, pour tout x dans T , on a:

$$u_x(f) = f(x) \left[\sum_{y \in T} p_{xy} u_y(f) + \sum_{y \in A} p_{xy} f(y) \right] \equiv \sum_{y \in S} p_{xy} u_y(f).$$

Preuve. On a:

$$\begin{aligned} u_x(f) &= E \left[\prod_{k=0}^{\eta} f(X_k) \mid X_0 = x \right] \\ &= f(x) E \left[\prod_{k=1}^{\eta} f(X_k) \mid X_0 = x \right] \\ &= f(x) \sum_y E \left[\prod_{k=1}^{\eta} f(X_k) \mid X_1 = y \right] P(X_1 = y \mid X_0 = x) \\ &= f(x) \left[\sum_{y \in T} u_y(f) p_{xy} + \sum_{y \in A} f(y) p_{xy} \right] \\ &= f(x) \sum_{y \in S} u_y(f) p_{xy}. \end{aligned}$$

Pour obtenir une estimation de θ , il suffit de simuler un arbre de coalescence en partant de la base (des n séquences de l'échantillon) jusqu'à l'ancêtre commun, en choisissant les événements en fonction des transitions déterminées. La vraisemblance est alors la moyenne sur tous les arbres simulés du produit des fonctions f pour chaque arbre.

Nous reviendrons en détail sur cette procédure d'estimation dans le chapitre 3, puisque c'est elle que nous utiliserons. Notons que la méthodologie a fait l'objet de nombreux travaux, comme ceux de Griffiths (1989) et Griffiths et Tavaré (1994a, 1994b, 1994c, 1996, 1997).

2.5.2. Amélioration de Fearnhead et Donnelly

Stephens et Donnelly (2000) ont travaillé à améliorer la méthode de Griffiths et Tavaré dans un cadre d'échantillonnage pondéré ("importance sampling"), en construisant une meilleure distribution proposée pour construire les arbres de

coalescence. Fearnhead et Donnelly (2001) étendent ces travaux dans le cas où la recombinaison est considérée. Notons cependant que l'on parle ici d'estimer un taux de recombinaison (ou de mutation) sur toute une séquence, et qu'il n'est pas question de cartographie génétique, contrairement à la méthode que l'on propose dans ce travail.

Soient n séquences dont la configuration est notée \mathbf{H}_0 (configuration qui contient la multiplicité de chaque type de séquence), et G une histoire ancestrale (ARG incluant les mutations et leurs types). Soit $P(G|\rho, \theta)$ la densité d'une histoire ancestrale G étant donnés les paramètres de recombinaison et de mutation ρ et θ . La vraisemblance des paramètres ρ et θ s'écrit:

$$L(\rho, \theta) = P(\mathbf{H}_0|\rho, \theta) = \int P(\mathbf{H}_0|G, \rho, \theta) P(G|\rho, \theta) dG,$$

où l'intégrale est évaluée sur toutes les histoires ancestrales de n branches, et $P(\mathbf{H}_0|G, \rho, \theta)$ est défini tel que:

$$P(\mathbf{H}_0|G, \rho, \theta) = \begin{cases} 1 & \text{si } G \text{ est compatible avec les données} \\ 0 & \text{sinon.} \end{cases}$$

Soit \bar{G} l'ensemble des histoires ancestrales compatibles avec les données. Soit $Q(G)$ une densité dont le support contient \bar{G} . On a alors:

$$L(\rho, \theta) = \int_{\bar{G}} \frac{P(G|\rho, \theta)}{Q(G)} Q(G) dG,$$

qui est estimé par:

$$\frac{1}{M} \sum_{i=1}^M \frac{P(G_i|\rho, \theta)}{Q(G_i)},$$

où $\{G_1, \dots, G_M\}$ est un échantillon selon $Q(G)$. L'expression ci-dessus est une approximation par échantillonnage pondéré de la vraisemblance, et $Q(G)$ est la distribution d'échantillonnage pondéré proposée. De plus, $P(G_i|\rho, \theta)/Q(G_i)$ est le poids d'échantillonnage pondéré. Soient les états successifs $\mathbf{H}_0, \mathbf{H}_1, \dots, \mathbf{H}_\tau$ en

remontant dans le temps jusqu'au MRCA, \mathbf{H}_i étant l'état des séquences ancestrales de l'échantillon après i transitions. En suivant Stephens et Donnelly (2000), la densité proposée optimale est:

$$Q(\mathbf{H}_{i+1}|\mathbf{H}_i) = P(H_i|\mathbf{H}_{i+1}) \frac{\pi(\mathbf{H}_{i+1})}{\pi(\mathbf{H}_i)},$$

où $P(\mathbf{H}_i|\mathbf{H}_{i+1})$ est la probabilité de transition de \mathbf{H}_{i+1} à \mathbf{H}_i (connue, car c'est la probabilité de transition en avant dans un processus de coalescence avec recombinaison, l'ARG), et $\pi(\mathbf{H}_i)$ est la probabilité qu'un échantillon de la population de même taille que \mathbf{H}_i donne la configuration \mathbf{H}_i .

Définition. Soit $\pi(\cdot|\mathbf{H})$ la distribution conditionnelle d'une séquence supplémentaire échantillonnée, sachant que les autres séquences échantillonnées sont de type \mathbf{H} :

$$\pi(\alpha|\mathbf{H}) = \pi(\{\mathbf{H}, \alpha\})/\pi(\mathbf{H}).$$

Il ne reste qu'à définir $\pi(\alpha|\mathbf{H})$ pour que la distribution proposée soit définie. Considérons d'abord l'approximation de $\pi(\alpha|\mathbf{H})$ quand $\rho = 0$. Quand \mathbf{H} contient j séquences, le nouveau type α est obtenu en choisissant une séquence de \mathbf{H} au hasard en le faisant muter un nombre géométrique de fois. Si j_β est le nombre de chromosomes de type β dans \mathbf{H} , alors (Stephens et Donnelly, 2000):

$$\hat{\pi}(\alpha|\mathbf{H}) = \sum_{\beta} \frac{j_{\beta}}{j} \sum_{k=0}^{\infty} \frac{j}{j+\theta} \left(\frac{\theta}{j+\theta} \right)^k P_{\beta\alpha}^k.$$

Stephens et Donnelly (2000) montrent que cette distribution possède d'intéressantes propriétés et que, dans certains cas simples, elle est même optimale. L'approximation de $\pi(\alpha|\mathbf{H})$ quand $\rho \neq 0$ se fait comme suit:

- (i) Initialisation: soit j le nombre de séquences dans H . Soit s le nombre de sites ségrégants aux positions x_1, \dots, x_s . Soit $y_i = (x_i + x_{i+1})/2$ les $s - 1$ points centraux successifs. Soit $z_i = x_{i+1} - x_i$.
- (ii) Recombinaison: pour $i = 1, \dots, s - 1$, générons une recombinaison au point y_i avec probabilité $z_i\rho/(z_i\rho + j)$, indépendamment des recombinaisons à y_l ,

- pour $l \neq i$. Soit k le nombre de recombinaisons générées. Ces k recombinaisons cassent la séquence en $k + 1$ intervalles $\{r_1, \dots, r_{k+1}\} = r$.
- (iii) Imputation: imputons les types aux sites ségrégants non ancestraux dans H . Si p_i est la proportion de séquences dans H qui sont de type i à un site spécifique, alors un type i est imputé à ce site avec probabilité p_i . Toutes les imputations sont faites indépendamment les unes des autres.
 - (iv) Mutation: on traite chaque r_i indépendamment. Pour chacun, on simule le type d'une nouvelle séquence de type α sur cette région selon l'approximation $\pi(\alpha|\mathbf{H})$ quand $\rho = 0$; il faut simuler le type d'une séquence complète selon l'approximation quand $\rho = 0$, et utiliser la valeur du type sur l'intervalle r_i . Le type de la séquence α est obtenue par l'union des types r_i sur les $k + 1$ intervalles.

L'approximation est une somme sur toutes les imputations et recombinaisons possibles. Fearnhead et Donnelly comparent ensuite leur méthode avec celle de Griffiths et Tavaré sur des exemples, et il semble qu'elle soit meilleure.

2.5.3. Autres méthodes

D'autres auteurs se sont penchés sur le problème de l'estimation de ρ , qui bien que différent du travail que l'on propose dans cette thèse, est relié au nôtre. Hey et Wakeley (1997), proposent un estimateur basé sur le processus de coalescence pour un échantillon de quatre séquences. Cet estimateur emploie des estimés à maximum de vraisemblance qui sont ensuite combinés pour des paires de sites informatifs et des sous-ensembles de quatre séquences. Kuhner *et al.* (2000) proposent une méthode MCMC (Markov Chain Monte Carlo) pour estimer ρ où la généalogie est basée sur l'ARG. Une généalogie est générée selon une probabilité *a priori* basée sur le processus de coalescence, et selon une valeur initiale pour le paramètre d'intérêt, puis l'algorithme MCMC va modifier la généalogie selon certaines distributions de probabilités afin d'en trouver de meilleures. Les auteurs rapportent que la méthode fonctionne bien, mais des résultats moins concluants sont rapportés dans Fearnhead et Donnelly (2001) quand ils comparent leur méthode à celle de Griffiths et Marjoram (1997) et à celle de Kuhner *et al.* (2000). Cette méthode est similaire à celle de Nielsen (2000) qui développe une méthode MCMC pour analyser des SNPs liés, dont l'histoire est décrite par

l'ARG pour une population de taille constante. Des estimations par une approche bayésienne sont obtenues. Plus récemment, Hudson considère la probabilité de paires de sites en fonction de leur distribution sous un modèle neutre, et montre comment utiliser cette distribution pour tester le déséquilibre de liaison (Hudson, 2001b). Pour plus de deux sites, un estimateur de vraisemblance basé sur une combinaison des distributions des paires de sites est développé. D'autres méthodes existent également, surtout des méthodes basées sur des combinaisons de statistiques descriptives. On trouvera une comparaison de ces méthodes dans Wall (2000). Aussi, Stephens (2001) propose une revue des différentes méthodes d'estimation pour le processus de coalescence.

Chapitre 3

Cartographie fine par le graphe de recombinaison ancestral

3.1. Introduction

Comme nous l'avons vu, le processus de coalescence offre une façon de modéliser la généalogie de séquences d'ADN sans considérer la recombinaison. Les événements de mutation et de coalescence peuvent être tous deux pris en compte et le résultat peut être représenté par un arbre. L'extension du processus de coalescence traditionnel, le graphe de recombinaison ancestral, tient compte des recombinaisons, et a été décrit par Griffiths et Marjoram (1996, 1997). Quand la recombinaison est incluse dans un processus de coalescence, une séquence peut provenir d'une ou de deux séquences parentales de la génération précédente (en fait deux si une recombinaison s'est produite, sinon, une seule), et nous devons

donc considérer un graphe au lieu d'un arbre. Notons que tenir compte de la recombinaison dans le but de modéliser l'évolution d'un ensemble de séquences est particulièrement important lorsque l'on tente de développer un outil de cartographie génétique car la recombinaison est le processus fondamental qui mélange l'ADN entre les séquences quand les chromosomes sont transmis de génération en génération.

Les applications du processus de coalescence ont été nombreuses, mais ont, jusqu'à maintenant, été essentiellement basées sur une suite de nucléotides, c'est-à-dire une séquence d'ADN. Pour des fins de cartographie, il nous faut considérer un autre type de séquence: une séquence de marqueurs, c'est-à-dire un ensemble ordonné de fragments d'ADN (marqueurs), parmi lesquels certains peuvent être variables dans la population, distants entre eux, de telle sorte que la probabilité d'une recombinaison entre deux marqueurs dépend de la distance qui les sépare.

Supposons une population constituée de $2N$ séquences qui évoluent selon un modèle de Wright-Fisher; en particulier, la taille de la population est supposée constante, les générations sont discrètes et ne se chevauchent pas, les accouplements se font au hasard. La généalogie de l'échantillon de séquences est modélisée en remontant le temps, en commençant par l'échantillon observé jusqu'à ce que l'ancêtre commun le plus récent (MRCA, "most recent common ancestor") de l'échantillon soit trouvé. À chaque changement dans le graphe, l'un des événements suivants peut se produire:

- 1) deux séquences coalescent si elles partagent un ancêtre commun;
- 2) une séquence mute et le matériel génétique à un marqueur donné dans une séquence est changé;
- 3) une séquence recombine.

Quand un événement de recombinaison se produit, un point de recombinaison est choisi au hasard selon une distribution donnée; la séquence est séparée en deux parties afin de créer les deux séquences parentales appelées les séquences parentales de "gauche" et de "droite". La séquence parentale de gauche

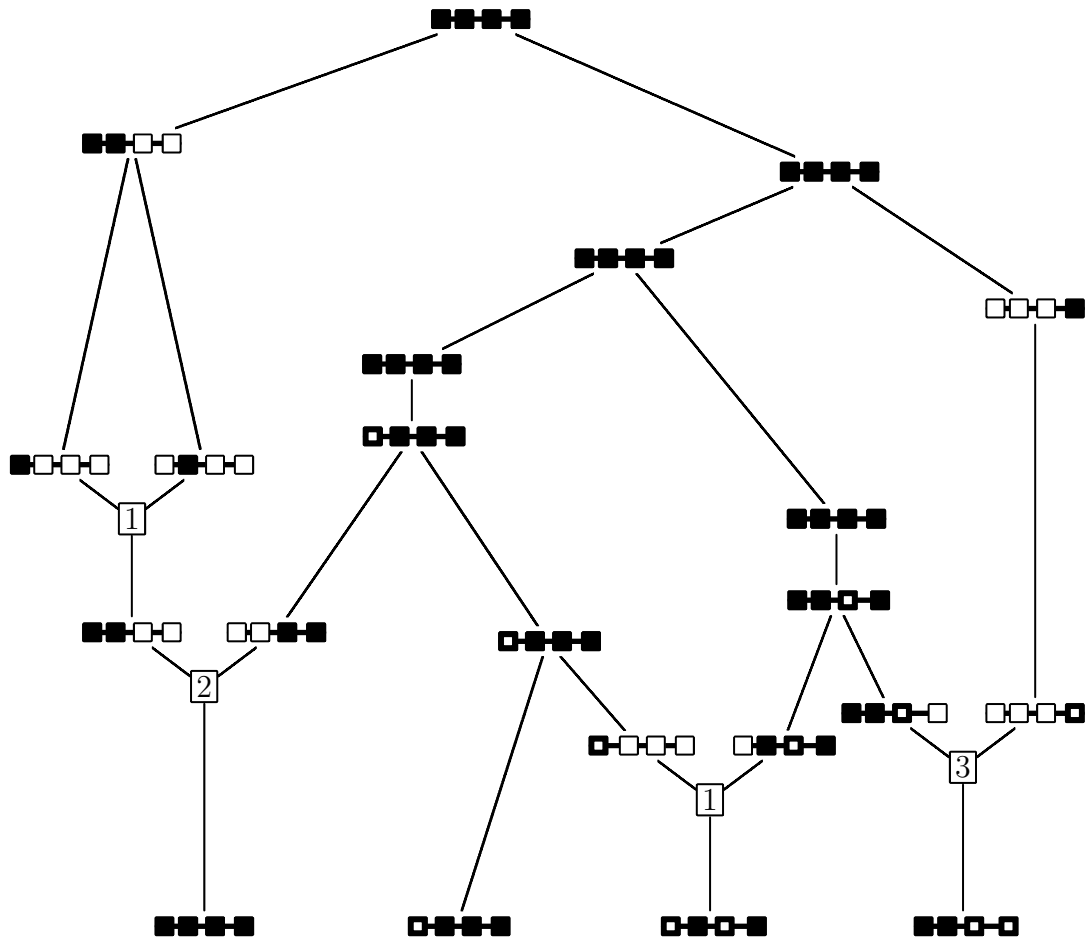


Figure 3.1.1. Un exemple d'ARG. Symboles: ■ marqueur ancestral primitif; ◼ marqueur ancestral mutant; □ marqueur non ancestral.

possède le même matériel génétique que “l’enfant” du début de la séquence jusqu’au point de recombinaison. La séquence parentale de droite a le même matériel génétique que l’enfant du point de recombinaison jusqu’à la fin de la séquence.

Le temps est mesuré en unités de $2N$ générations, et N est supposé grand. Le taux de mutation par séquence par génération est u et on définit $\theta = 4Nu$. De la même façon, on définit $\rho = 4Nr$, où r est le taux de recombinaison par séquence par génération. Les paramètres θ et ρ sont supposés constants alors que l’on fait tendre N vers l’infini. Alors, le nombre de séquences ancestrales dans le graphe est un processus de naissance et de mort avec taux de naissance et de

mort respectifs $k\rho/2$ et $k(k-1)/2$, quand le nombre actuel de séquences est k . Un événement de coalescence diminue le nombre d'ancêtres de un, alors qu'un événement de mutation ne change pas le nombre d'ancêtres, et qu'un événement de recombinaison augmente le nombre d'ancêtres de un. Comme les événements de coalescence se produisent à un taux quadratique et les recombinaisons à un taux linéaire, le nombre d'ancêtres reste fini, et le graphe conduira éventuellement à un seul ancêtre, le MRCA de l'échantillon.

Un exemple de graphe de recombinaison ancestral, adapté de l'article de Griffiths et Marjoram (1996), est présenté à la figure 3.1.1. Ce graphe montre explicitement le matériel ancestral et non ancestral des séquences. En reculant dans le temps, un événement de coalescence se produit quand deux séquences se rejoignent pour n'en former qu'une; un événement de mutation se produit quand un allèle d'un marqueur dans une séquence change; et un événement de recombinaison se produit quand une séquence se sépare en deux (le numéro de l'intervalle où la recombinaison se produit est indiqué dans une petite boîte sur la figure 3.1.1).

3.2. Le modèle

On considère des séquences de longueur suffisamment petite, de façon telle que l'on puisse ignorer l'interférence et que les distances, mesurées en Morgans, puissent être considérées additives. À chaque génération, chaque séquence provient de deux séquences parentales avec probabilité r et d'une seule séquence parentale avec probabilité $1-r$. Nous supposons qu'il n'y a pas d'interférence et que r est la distance entre un premier marqueur au début de la séquence et un dernier à la fin.

On considère un échantillon de séquences chez des cas et des témoins qui, respectivement, expriment (cas) et n'expriment pas (témoins) un certain phénotype. Le caractère (i.e., la maladie) est supposé être causé par une seule mutation, et cette mutation ne s'est produite qu'une seule fois dans l'histoire (ceci est connu

sous le nom de “infinitely many site model”). De plus, l'échantillon provient d'une jeune population isolée, et la maladie qui crée la mutation a une haute pénétrance. Nos modèles et méthodes sont destinées à identifier la position de cette mutation.

Une séquence est composée de L marqueurs. Ceux-ci sont ordonnés, du premier qui marque le début de la séquence jusqu'au dernier qui en marque la fin, et les positions exactes de $L - 1$ d'entre eux sont connues. On parlera du locus m pour désigner la position du marqueur m dans la séquence. On dénote par r_m la distance en Morgans entre les loci m et $m + 1$. La mutation qui influence le caractère (TIM, “trait-influencing mutation”) est elle-même un marqueur à l'intérieur de la séquence, mais sa position est inconnue. Les allèles de ce marqueur peuvent être inférés à partir du phénotype. Nous ferons la distinction entre des intervalles entre marqueurs (au nombre de $L - 1$) et entre marqueurs de position connue (au nombre de $L - 2$). Le TIM est traité de la même façon que les autres marqueurs, dans la construction de la généalogie. Nous supposons que le TIM est entre le premier et le dernier marqueur, et qu'il ne peut donc pas être à l'extérieur de la séquence. De plus, le TIM est supposé être à une distance r_T du premier marqueur de la séquence. Nous cherchons un estimateur de r_T par maximum de vraisemblance.

Nous supposons que les mutations sont rares et non récurrentes, c'est-à-dire qu'au plus un événement de mutation se produit par marqueur dans l'histoire des séquences. La conséquence de ceci est qu'une mutation n'est possible sur une séquence à un marqueur donné que si aucune autre séquence ne possède un allèle mutant à ce locus, et que cette séquence est unique au moment de l'événement, c'est-à-dire qu'elle a une multiplicité de 1. Le taux de mutation étant petit, nous considérons au plus deux allèles par marqueur. De plus, le taux de mutation est identique à chaque marqueur. Notons qu'il est aisé de changer ces hypothèses, de supposer des mutations récurrentes et des taux de mutation variables à chaque marqueur. Cependant, ces hypothèses facilitent grandement la reconstruction des

généalogies, et sont de plus tout à fait réalistes si l'on travaille avec des marqueurs à faible taux de mutation, comme des SNPs.

Soit r la longueur totale de la séquence, c'est-à-dire $r = \sum_{p=1}^{L-1} r_p$. Soit x_m la position du marqueur m , en prenant pour convention que le premier marqueur est à l'origine, c'est-à-dire:

$$\begin{cases} x_1 = 0 \\ x_m = \sum_{p=1}^{m-1} r_p, & 2 \leq m \leq L. \end{cases}$$

On peut effectuer une partition des séquences en intervalles, où l'intervalle p est le segment entre les marqueurs p et $p + 1$. Une séquence est illustrée dans la figure 3.2.1. On comprend qu'une "séquence" est simplement une suite ordonnée de marqueurs à des positions connues, à l'exception du TIM dont on cherche la position.

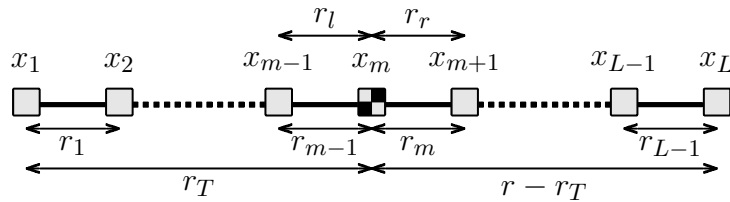
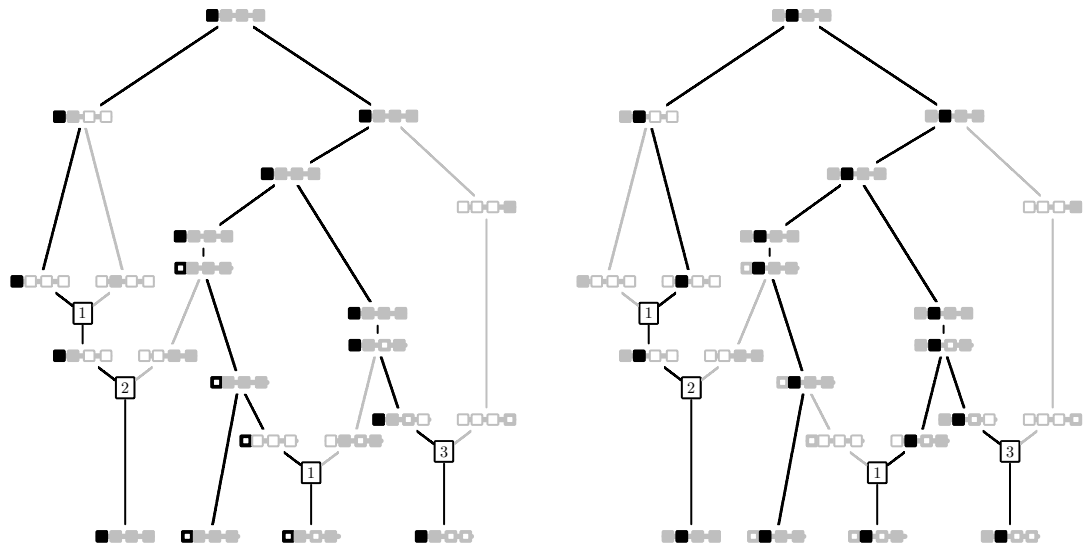


Figure 3.2.1. Configuration d'une séquence si le TIM (■) est à la position m . On définit par r_l la distance entre le TIM et le marqueur immédiatement à gauche, et r_r la distance entre le TIM et le marqueur immédiatement à droite.

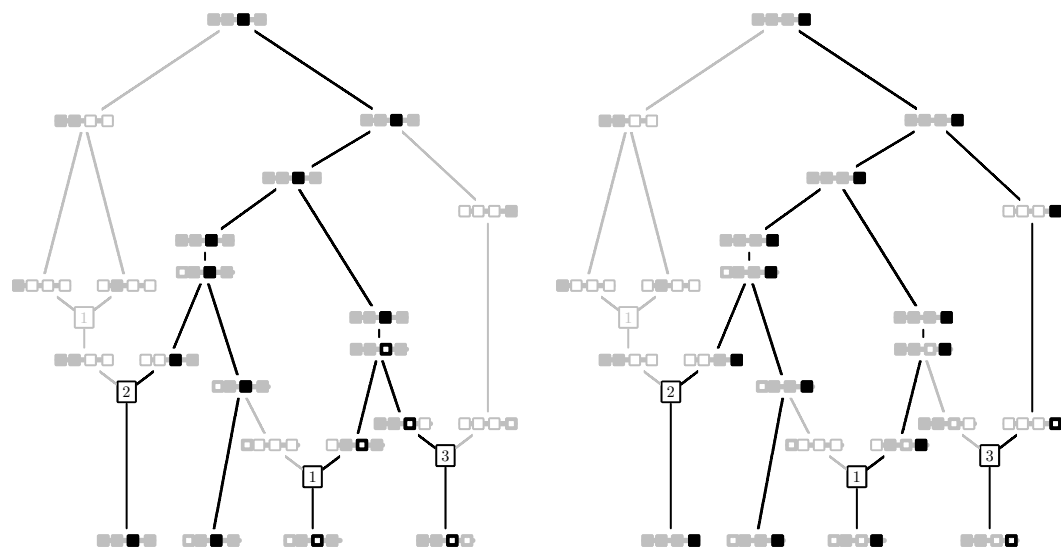
Chaque marqueur m possède un arbre de coalescence $T(m)$ décrivant l'histoire de l'échantillon à ce locus. Pour obtenir $T(m)$, partons de l'échantillon initial, et pour chaque séquence, suivons les arêtes du graphe de recombinaison ancestral au marqueur m ; quand un événement de recombinaison se produit, prenons le chemin de gauche si le point de recombinaison est après m , sinon prenons le chemin de droite. L'ensemble de ces arêtes définit $T(m)$. La figure 3.2.2 illustre ce concept en montrant les arbres partiels pour le graphe de recombinaison ancestral de la figure 3.1.1.

Un marqueur m dans une séquence donnée est ancestral si ce marqueur est inclus dans $T(m)$, sinon, il est non ancestral. De plus, un marqueur ancestral peut être de deux types différents: un type “primitif” (■), s’il s’agit d’une copie de l’allèle du MRCA sans mutation, ou un type mutant (◻), si c’est une copie



(i) Arbre partiel pour le premier marqueur.

(ii) Arbre partiel pour le second marqueur.



(iii) Arbre partiel pour le troisième marqueur. (iv) Arbre partiel pour le quatrième marqueur.

Figure 3.2.2. Arbres de coalescence partiels pour chacun des quatre marqueurs du graphe de recombinaison ancestral de la figure 3.1.1.

mutante d'une allèle du MRCA. Un marqueur non ancestral sera représenté par le symbole \square .

Le $\tau^{\text{ième}}$ événement de coalescence, mutation ou recombinaison, affectant le matériel ancestral des séquences échantillonnées aux L loci considérés en reculant dans le temps ($\tau = 0, \dots, \tau^*$, où 0 correspond à l'échantillon initial et τ^* à la dernière coalescence au MRCA) se produit au temps t_τ . Dénotons par \mathbf{H}_τ l'ensemble des séquences ancestrales au temps t_τ juste après que le $\tau^{\text{ième}}$ événement se soit produit. En fait, \mathbf{H}_τ est un ensemble de séquences $\mathbf{s} = (s_1, \dots, s_L)$ où s_m est l'allèle, primitif ou mutant, au marqueur m s'il est ancestral, ou un allèle quelconque indistingué au marqueur m s'il est non ancestral, pour $m = 1, \dots, L$. Nous considérons seulement les séquences qui possèdent au moins un marqueur ancestral. La notation suivante est utilisée pour les événements possibles qui se produisent en remontant dans le temps:

$$\begin{cases} C_i & \text{Co de deux séquences identiques } i \\ C_{ij}^k & \text{Co des séquences } i \text{ et } j \text{ vers la séquence } k \\ M_i^j(m) & \text{Mu de la séquence } i \text{ vers la séquence } j \text{ au marqueur } m \\ R_i^{jk}(p) & \text{Re de la séquence } i \text{ dans l'intervalle } p \text{ vers les séquences } j \text{ et } k \end{cases}$$

où Co signifie coalescence, Mu mutation et Re recombinaison.

Soit $Q(\mathbf{H}_\tau)$ la distribution de probabilité de l'état \mathbf{H}_τ . Alors, $Q(\mathbf{H}_\tau)$ est une fonction de $Q(\mathbf{H}_{\tau+1})$. On écrira par convention, $\mathbf{H}_{\tau+1} = \mathbf{H}_\tau + R_i^{jk}(p)$, si \mathbf{H}_τ est modifié par un événement de recombinaison $R_i^{jk}(p)$ au temps $t_{\tau+1}$, et ainsi de suite.

Si une recombinaison se produit, la densité $f_Z(z)$ du point de recombinaison est:

$$f_Z(z) = \frac{1}{r}, \quad \text{if } 0 < z < r.$$

Notons que n'importe quelle distribution pourrait être utilisée pour cette densité, par exemple pour tenir compte des régions où l'on sait que les recombinaisons sont plus denses ("hot spots"), ou encore des haplotypes en bloc (Gabriel

et al., 2002). En effet, certaines recherches récentes démontrent que des bouts de séquences sont transmis de génération en génération en bloc: dans ces blocs, peu ou pas de recombinaisons sont observées. Il est possible de modéliser ces phénomènes par la méthode en choisissant une distribution appropriée des points de recombinaison. Même si un point de recombinaison peut être situé n'importe où dans la séquence, seulement un événement de recombinaison entre deux marqueurs ancestraux affectera l'échantillon ancestral. Nous nommerons de telles recombinaisons des “recombinaisons ancestrales”, comme dans Fearnhead et Donnelly (2001).

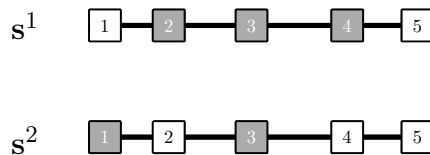


Figure 3.2.3. Exemple de deux séquences s^1 et s^2 de cinq marqueurs numérotés $1, \dots, 5$. (■ représente un marqueur ancestral, □ un marqueur non ancestral).

Considérons, par exemple, les deux séquences s^1 et s^2 de cinq marqueurs de la figure 3.2.3. En consultant cette figure, nous observons que si un événement de recombinaison se produit entre les loci 1 et 2 de s^1 , les deux séquences parentales seront d'un côté une séquence de marqueurs non ancestraux, et de l'autre côté la même séquence que la séquence originale. Cela illustre le fait que cet événement ne modifie pas l'histoire de l'échantillon et qu'il n'est pas informatif quant à l'histoire de celui-ci. Soit γ_i (κ_i) le rang du premier (dernier) intervalle d'une séquence de type i où un événement de recombinaison peut affecter le matériel ancestral. Par exemple, pour la figure 3.2.3, $\gamma_1 = 2$, $\kappa_1 = 3$ pour la séquence s^1 , et $\gamma_2 = 1$, $\kappa_2 = 2$ pour la séquence s^2 . De plus, soit:

$$c_i = \frac{1}{r} \left[\max\{x_m; \text{marqueur } m \in A_i\} - \min\{x_m; \text{marqueur } m \in A_i\} \right],$$

où A_i représente l'ensemble des marqueurs ancestraux de la séquence i . Alors, c_i

représente la proportion de la séquence i pour laquelle un événement de recombinaison peut affecter le matériel ancestral. De plus, soit:

$$b = \sum_{i=1}^d n_i (\max\{x_m; \text{marqueur } m \in A_i\} - \min\{x_m; \text{marqueur } m \in A_i\}).$$

Alors, $0 \leq b \leq nr$ et b est la longueur totale sur toutes les séquences où un événement de recombinaison peut affecter le matériel ancestral dans \mathbf{H}_τ , en prenant en compte les multiplicités des séquences.

De façon similaire, un événement de mutation n'affecte pas nécessairement l'histoire de l'échantillon. C'est le cas seulement s'il y a mutation à un marqueur ancestral. Dénotons par $|A_i|$ le nombre de marqueurs ancestraux de la séquence i où un événement de mutation affecte l'histoire de l'échantillon, et par $a = \sum_{i=1}^d n_i |A_i|$, le nombre total correspondant de tels marqueurs sur toutes les séquences de \mathbf{H}_τ ($n \leq a \leq nL$). Dans la figure 3.2.3 par exemple, si la multiplicité des séquences \mathbf{s}^1 et \mathbf{s}^2 est 1, alors $a = 3 + 2$. Notons que les termes a et b qui viennent d'être définis dépendent en fait de τ , l'étape à laquelle le graphe se trouve, bien qu'ils ne soient pas indicés par τ , afin d'alléger la notation.

Une coalescence de séquences de différents types i et j est possible, si celles-ci ne possèdent pas un ensemble de mutations incompatibles; cet événement est donc possible si, pour aucun marqueur m ($m = 1, \dots, L$), la séquence i (j) ne possède pas un allèle ancestral mutant au marqueur m et que la séquence j (i) ne possède pas un allèle ancestral primitif au même marqueur m . Ainsi, les séquences \mathbf{s}^1 et \mathbf{s}^3 de la figure 3.2.4 ne peuvent pas coalescer, car la séquence \mathbf{s}^1 possède un allèle ancestral primitif au troisième marqueur, alors que la séquence \mathbf{s}^3 possède un allèle ancestral mutant à ce marqueur.

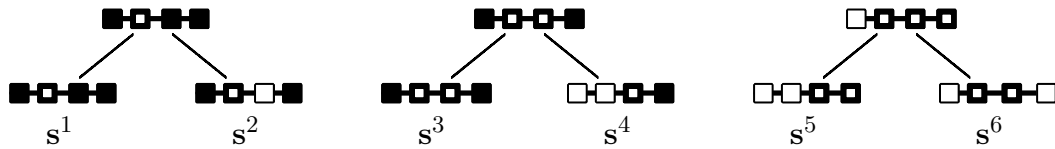


Figure 3.2.4. Exemples de coalescences possibles entre séquences de différents types (■ représente un site ancestral primitif, ◼ représente un site ancestral mutant, et □ un site non ancestral).

En notant $\mathbf{s}^i = (s_1^i, \dots, s_L^i)$ la séquence i , où s_m^i est l'allèle au marqueur m de la séquence i , la séquence résultante k (\mathbf{s}^k) d'une coalescence des séquences de différents types i et j est définie telle que :

$$\begin{aligned} s_m^k &= \blacksquare, & \text{si } s_m^i &= \blacksquare \text{ ou } s_m^j &= \blacksquare, \\ s_m^k &= \blacksquare, & \text{si } s_m^i &= \blacksquare \text{ ou } s_m^j &= \blacksquare, \\ s_m^k &= \square, & \text{si } s_m^i &= \square \text{ et } s_m^j &= \square, \end{aligned}$$

où $1 \leq m \leq L$, \blacksquare représente un site ancestral primitif, \blacksquare représente un site ancestral mutant, et \square un site non ancestral. Trois exemples de coalescence de séquences différentes sont présentés dans la figure 3.2.4.

3.3. Définition des probabilités de récurrence

Considérons n séquences ancestrales dont n_i de type i pour $i = 1, \dots, d$. Comme nous l'avons vu, lorsqu'il y a coalescence, mutation, ou recombinaison en reculant dans le temps, ces événements se produisent avec les probabilités :

$$\begin{cases} n(n-1)/[n(n-1) + n\theta + n\rho] & \text{pour une coalescence} \\ n\theta/[n(n-1) + n\theta + n\rho] & \text{pour une mutation} \\ n\rho/[n(n-1) + n\theta + n\rho] & \text{pour une recombinaison} \end{cases}$$

Si un événement de coalescence se produit, il y a $n-1$ séquences un pas en arrière dans le temps. Deux séquences de même type i vont donc coalescer avec probabilité $(n_i - 1)/(n-1)$, car un pas en arrière dans le temps, il y a $n_i - 1$ séquences de type i . Si deux séquences i et j de types différents, mais compatibles en ce qui a trait au matériel ancestral, coalescent vers une séquence k dont le type peut déjà exister ou non, nous devons tenir compte du fait que les types i et k , ou j et k , peuvent être identiques. Alors, la probabilité que i et j coalescent vers k est $(n_k + 1 - \delta_{ik} - \delta_{jk})/(n-1)$, où $\delta_{ik} = 1$ si $i = k$, et 0 si $i \neq k$.

Si le premier événement en reculant dans le temps est un événement de mutation, alors il y a une séquence i qui provient d'une séquence j , qui peut déjà exister ou non, avec probabilité $(n_j + 1)/n$, car un pas en arrière dans le

temps le nombre de séquences j est $n_j + 1$ (le nombre actuel de séquences j , plus la nouvelle produite par l'événement), et le nombre total de séquences reste inchangé à n . Une telle mutation est possible si le marqueur est ancestral, donc si m appartient à A_i , et si l'allèle mutant est unique, c'est à dire si $s_m^i \neq s_m^k$ pour tout $k \neq i$. Comme le taux de mutation θ est défini sur la séquence au complet, il faut multiplier $(n_j + 1)/n$ par $1/L$ (nous avons supposé que le taux de mutation est identique pour chaque marqueur) afin de tenir compte de la probabilité de mutation à un marqueur donné. De plus, un événement de mutation se produit dans du matériel non ancestral avec probabilité $(nL - a)/nL$.

Un événement de recombinaison peut se produire dans n'importe quel des $L - 1$ intervalles entre loci de la séquence i , et cela pour chaque séquence $i = 1, \dots, d$. Cependant, une recombinaison n'est ancestrale que si celle-ci se produit entre γ_i et κ_i . Quand cela se produit dans un intervalle donné d'une séquence i , que nous noterons p , alors cette séquence i a le même matériel génétique qu'une séquence j à gauche du point de recombinaison (p), ainsi que le même matériel génétique qu'une séquence k à droite du point de recombinaison (p). Dans ce contexte, nous considérons donc des paires de séquences ordonnées j et k . Un pas en arrière dans le temps, on a $n_j + 1$ séquences de type j et $n_k + 1$ séquences de type k . Comme le nombre total de séquences est alors $n + 1$, le nombre total de paires ordonnées de séquences possibles est $n(n + 1)$. De plus, la probabilité que la recombinaison se produise dans un certain intervalle est proportionnelle à la longueur de cet intervalle, donc la probabilité que la recombinaison ait lieu dans l'intervalle p est r_p/r . L'événement de recombinaison $R_i^{jk}(p)$ d'une séquence i dans l'intervalle p à partir des séquences j et k , qui peuvent exister ou non, a donc probabilité $[r_p/r][(n_j + 1)(n_k + 1)]/[n(n + 1)]$. De plus, comme $0 \leq b \leq nr$, un événement de recombinaison peut se produire sans affecter le matériel ancestral avec probabilité $(nr - b)/nr$.

Ces faits mènent à l'équation de récurrence suivante, quand le temps est mesuré en unités de $2N$ générations, analogue à celle discutée par Griffiths et

Marjoram (1996) dans la dérivation de leur formule pour un modèle de séquences continues:

$$\begin{aligned}
Q(\mathbf{H}_\tau) &= \frac{n(n-1)}{[n(n-1) + n\theta + n\rho]} \left[\sum_{\substack{i, \\ n_i > 1}} \frac{n_i - 1}{n-1} Q(\mathbf{H}_\tau + C_i) \right. & \textcircled{1} \\
&\quad \left. + 2 \sum_{\substack{i \neq j \\ \text{compatibles}}} \frac{n_k + 1 - \delta_{ik} - \delta_{jk}}{n-1} Q(\mathbf{H}_\tau + C_{ij}^k) \right] & \textcircled{2} \\
&+ \frac{n\theta}{[n(n-1) + n\theta + n\rho]} \left[\sum_i \sum_{\substack{m \in A_i, s_m^i \neq s_m^j \\ \text{pour tout } k \neq i}} \frac{1}{L} \frac{n_j + 1}{n} Q(\mathbf{H}_\tau + M_i^j(m)) \right. & \textcircled{3} \\
&\quad \left. + \frac{(nL - a)}{nL} Q(\mathbf{H}_\tau) \right] & \textcircled{4} \quad (3.1) \\
&+ \frac{n\rho}{[n(n-1) + n\theta + n\rho]} \left[\sum_i \sum_{p=\gamma_i}^{\kappa_i} \frac{r_p}{r} \frac{(n_j + 1)(n_k + 1)}{n(n+1)} Q(\mathbf{H}_\tau + R_i^{jk}(p)) \right. & \textcircled{5} \\
&\quad \left. + \frac{(nr - b)}{nr} Q(\mathbf{H}_\tau) \right], & \textcircled{6}
\end{aligned}$$

où les nombres à droite des termes réfèrent aux événements suivants:

- ① coalescence de deux séquences du même type i ;
- ② coalescence de deux séquences de types différents, i et j , vers une séquence de type k ;
- ③ mutation d'une séquence i au marqueur m à partir d'une séquence j , où la séquence j peut exister ou non;
- ④ mutation dans du matériel non ancestral;
- ⑤ recombinaison d'une séquence i dans l'intervalle p à partir des séquences j et k , où les séquences j et k peuvent exister ou non;
- ⑥ recombinaison dans du matériel non ancestral.

Notons que le paramètre r_T n'apparaît pas dans la formule précédente, ce qui est troublant compte tenu que c'est précisément ce paramètre que l'on cherche à estimer. En fait, pour deux valeurs de p , correspondant aux intervalles de chaque coté du TIM, les valeurs de r_p dépendent de la valeur de r_T . Nous allons maintenant montrer que $Q(\mathbf{H}_\tau)$ dépend bien de r_T . Soient r_l et r_r les longueurs

des intervalles à la gauche et à droite du TIM qui dépendent seulement de r_T .

Définissons:

$$\delta_p^l = \begin{cases} 1 & \text{si } x_{p+1} = r_T \\ 0 & \text{sinon} \end{cases}$$

$$\delta_p^r = \begin{cases} 1 & \text{si } x_p = r_T \\ 0 & \text{sinon} \end{cases}$$

En d'autres termes, δ_p^l (δ_p^r) est égal à 1 si le TIM est le marqueur à droite (à gauche) de l'intervalle p , et égal à 0 sinon. Avec cette notation, le terme de recombinaison (⑤) dans la formule précédente peut s'écrire sous la forme:

$$\sum_i \sum_{p=\gamma_i}^{\kappa_i} \left[\frac{r_p}{r} (1 - \delta_p^l)(1 - \delta_p^r) + \frac{r_l}{r} \delta_p^l + \frac{r_r}{r} \delta_p^r \right]$$

$$\times \frac{(n_j + 1)(n_k + 1)}{n(n + 1)} Q \left(\mathbf{H}_\tau + R_i^{jk}(p) \right).$$

Notons que $n_j + 1$ et $n_k + 1$ sont les nombres de séquences de types j et k , respectivement, existant un pas en arrière dans le temps. Dans le modèle de Griffiths et Marjoram (1996), en supposant une séquence continue, nous avons toujours $n_j = n_k = 0$; mais dans l'établissement du modèle pour une séquence que l'on considère comme une suite discrète de loci, n_j et n_k ne sont pas toujours nuls.

3.4. Identités algébriques et échantillonnage pondéré

Nous présentons maintenant la procédure de Monte Carlo proposée pour la première fois dans un tel contexte par Griffiths et Tavaré (1994c) que nous avons introduite dans la section 2.5.1.

Après quelques simplifications, une expression équivalente à (3.1) est:

$$Q(\mathbf{H}_\tau) = \frac{n\theta - a\theta/L + n\rho - b\rho/r}{[n(n-1) + n\theta + n\rho]} Q(\mathbf{H}_\tau)$$

$$+ \frac{n(n-1)}{[n(n-1) + n\theta + n\rho]} \sum_{\substack{i, \\ n_i > 1}} \frac{n_i - 1}{n - 1} Q(\mathbf{H}_\tau + C_i)$$

$$\begin{aligned}
& + \frac{2n(n-1)}{[n(n-1) + n\theta + n\rho]} \sum_{\substack{i \neq j \\ \text{compatibles}}} \frac{n_k + 1 - \delta_{ik} - \delta_{jk}}{n-1} Q(\mathbf{H}_\tau + C_{ij}^k) \\
& + \frac{n\theta}{[n(n-1) + n\theta + n\rho]} \sum_i \sum_{\substack{m \in A_i, s_m^i \neq s_m^j \\ \text{pour tout } k \neq i}} \frac{1}{L} \frac{n_j + 1}{n} Q(\mathbf{H}_\tau + M_i^j) \\
& + \frac{n\rho}{[n(n-1) + n\theta + n\rho]} \sum_i \left[\sum_{p=\gamma_i}^{\kappa_i} \frac{r_p}{r} \frac{(n_j + 1)(n_k + 1)}{n(n+1)} Q(\mathbf{H}_\tau + R_i^{jk}(p)) \right].
\end{aligned}$$

Soient $\alpha = a/(nL)$, $\beta = b/(nr)$, $\rho_p = 4Nr_p$,

$$D_{\mathbf{H}_\tau} = [n(n-1) + n\alpha\theta + n\beta\rho],$$

et

$$\begin{aligned}
S_{\mathbf{H}_\tau} &= n \sum_{\substack{i, \\ n_i > 1}} (n_i - 1) + 2n \sum_{\substack{i \neq j \\ \text{compatibles}}} (n_k + 1 - \delta_{ik} - \delta_{jk}) \\
& + \sum_i \sum_{\substack{m \in A_i, s_m^i \neq s_m^j \\ \text{pour tout } k \neq i}} \theta(n_j + 1)/L + \sum_i \sum_{p=\gamma_i}^{\kappa_i} [\rho_p(n_j + 1)(n_k + 1)]/[(n+1)].
\end{aligned}$$

Après quelques simplifications, l'équation de récurrence (3.1) peut alors s'écrire sous la forme:

$$\begin{aligned}
Q(\mathbf{H}_\tau) &= \sum_{\substack{i, \\ n_i > 1}} \frac{S_{\mathbf{H}_\tau}}{D_{\mathbf{H}_\tau}} \frac{n(n_i - 1)}{S_{\mathbf{H}_\tau}} Q(\mathbf{H}_\tau + C_i) \\
& + \sum_{\substack{i \neq j \\ \text{compatibles}}} \frac{S_{\mathbf{H}_\tau}}{D_{\mathbf{H}_\tau}} \frac{2n(n_k + 1 - \delta_{ik} - \delta_{jk})}{S_{\mathbf{H}_\tau}} Q(\mathbf{H}_\tau + C_{ij}^k) \\
& + \sum_i \sum_{\substack{m \in A_i, s_m^i \neq s_m^j \\ \text{pour tout } k \neq i}} \frac{S_{\mathbf{H}_\tau}}{D_{\mathbf{H}_\tau}} \frac{\theta(n_j + 1)}{LS_{\mathbf{H}_\tau}} Q(\mathbf{H}_\tau + M_i^j(m)) \\
& + \sum_i \sum_{p=\gamma_i}^{\kappa_i} \frac{S_{\mathbf{H}_\tau}}{D_{\mathbf{H}_\tau}} \frac{\rho_p(n_j + 1)(n_k + 1)}{S_{\mathbf{H}_\tau}(n+1)} Q(\mathbf{H}_\tau + R_i^{jk}(p)).
\end{aligned}$$

Définissons maintenant une chaîne de Markov avec probabilités de transition de \mathbf{H}_τ à $\mathbf{H}_{\tau+1}$, dénotées par $P(\mathbf{H}_{\tau+1}|\mathbf{H}_\tau)$. Au temps d'arrivée du $(\tau + 1)^{\text{ième}}$ événement, une transition est faite de \mathbf{H}_τ à:

$$\begin{aligned}
(\mathbf{H}_\tau + C_i) & \text{ avec probabilité } n(n_i - 1)/S_{\mathbf{H}_\tau}, \\
(\mathbf{H}_\tau + C_{ij}^k) & \text{ avec probabilité } 2n(n_k + 1 - \delta_{ik} - \delta_{jk})/S_{\mathbf{H}_\tau}, \\
(\mathbf{H}_\tau + M_i^j(m)) & \text{ avec probabilité } \theta(n_j + 1)/(LS_{\mathbf{H}_\tau}), \\
(\mathbf{H}_\tau + R_i^{jk}(p)) & \text{ avec probabilité } \rho_p(n_j + 1)(n_k + 1)/[(n + 1)S_{\mathbf{H}_\tau}].
\end{aligned} \tag{3.2}$$

On reconstruit ainsi un graphe de recombinaison ancestral. Soit:

$$f(\mathbf{H}_\tau, \mathbf{H}_{\tau+1}) = S_{\mathbf{H}_\tau}/D_{\mathbf{H}_\tau}. \tag{3.3}$$

L'état \mathbf{H}_τ est l'état du matériel ancestral suite au $\tau^{\text{ième}}$ événement de coalescence, mutation ou recombinaison pour $\tau = 0, \dots, \tau^*$ (où τ^* est le temps d'arrêt). Le temps d'arrêt est atteint lorsque le plus récent ancêtre commun (MRCA) est trouvé; la configuration est alors représentée par \mathbf{H}_{τ^*} . Comme le MRCA a des loci ancestraux primitifs partout, $\mathbf{H}(\tau^*)$ est déterminé uniquement avec probabilité 1. Cela signifie que $Q(\mathbf{H}_{\tau^*})$ est 1 pour une seule séquence, et 0 pour les autres. Nous avons alors:

$$Q(\mathbf{H}_\tau) = \sum_{\mathbf{H}_{\tau+1}} f(\mathbf{H}_\tau, \mathbf{H}_{\tau+1})P(\mathbf{H}_{\tau+1}|\mathbf{H}_\tau)Q(\mathbf{H}_{\tau+1}),$$

pour $\tau = 0, \dots, \tau^*$, et par récurrence:

$$\begin{aligned}
Q(\mathbf{H}_0) &= \sum_{\mathbf{H}_1} \sum_{\mathbf{H}_2} \cdots \sum_{\mathbf{H}_{\tau^*}} f(\mathbf{H}_0, \mathbf{H}_1)f(\mathbf{H}_1, \mathbf{H}_2)f(\mathbf{H}_2, \mathbf{H}_3) \cdots f(\mathbf{H}_{\tau^*-1}, \mathbf{H}_{\tau^*}) \\
&\times P(\mathbf{H}_1|\mathbf{H}_0)P(\mathbf{H}_2|\mathbf{H}_1) \cdots P(\mathbf{H}_{\tau^*}|\mathbf{H}_{\tau^*-1})Q(\mathbf{H}_{\tau^*}).
\end{aligned}$$

Donc, on a:

$$L(r_T) = Q(\mathbf{H}_0) = E_P \left[\prod_{\tau=0}^{\tau^*-1} f(\mathbf{H}_\tau, \mathbf{H}_{\tau+1}) \right].$$

Il s'agit d'une représentation d'échantillonnage pondéré avec P pour distribution proposée.

Soit $\Theta = \{\theta, r_T\}$ l'ensemble des paramètres inconnus du processus. Alors, sachant $\Theta_0 = \{\theta_0, r_{T_0}\}$, un estimé de $Q_\Theta(\mathbf{H}_0)$ peut être obtenu pour différentes valeurs de Θ . En effet, on a :

$$Q_\Theta(\mathbf{H}_\tau) = \sum_{\mathbf{H}_{\tau+1}} h_{\Theta\Theta_0}(\mathbf{H}_\tau, \mathbf{H}_{\tau+1}) P_{\Theta_0}(\mathbf{H}_{\tau+1}|\mathbf{H}_\tau) Q_\Theta(\mathbf{H}_{\tau+1}), \quad (3.4)$$

où

$$h_{\Theta\Theta_0}(\mathbf{H}_\tau, \mathbf{H}_{\tau+1}) = \frac{f_\Theta(\mathbf{H}_\tau, \mathbf{H}_{\tau+1}) P_\Theta(\mathbf{H}_{\tau+1}|\mathbf{H}_\tau)}{P_{\Theta_0}(\mathbf{H}_{\tau+1}|\mathbf{H}_\tau)}.$$

Donc $Q_\Theta(\mathbf{H}_0)$ peut être estimé par :

$$E_{P_{\Theta_0}} \left[\prod_{\tau=0}^{\tau^*-1} h_{\Theta\Theta_0}(\mathbf{H}_\tau, \mathbf{H}_{\tau+1}) \right].$$

En dénotant par r_{0l} et r_{0r} les paramètres r_l et r_r sous Θ_0 , et en utilisant f_Θ tel que défini par (3.3) et P_Θ défini par (3.2) sous la même hypothèse Θ_0 , la fonction $h_{\Theta\Theta_0}$ prend la forme :

$$h_{\Theta\Theta_0}(\mathbf{H}_\tau, \mathbf{H}_{\tau+1}) = \frac{S(\mathbf{H}_\tau, \Theta_0)}{D(\mathbf{H}_\tau, \Theta)} \phi(\mathbf{H}_\tau, \mathbf{H}_{\tau+1}),$$

où

$$\phi(\mathbf{H}_\tau, \mathbf{H}_{\tau+1}) = \begin{cases} \theta/\theta_0 & \text{si } Mu \\ \frac{r_p(1-\delta_p^l)(1-\delta_p^r)+r_l\delta_p^l+r_r\delta_p^r}{r_p(1-\delta_p^l)(1-\delta_p^r)+r_{0l}\delta_p^l+r_{0r}\delta_p^r} & \text{si } Re \\ 1 & \text{si } Co. \end{cases}$$

Cela va nous permettre de simuler des graphes avec une valeur conductrice Θ_0 , et d'obtenir la vraisemblance pour d'autres valeurs, c'est-à-dire d'obtenir une courbe de vraisemblance à partir d'une seule valeur conductrice.

Il est important de noter que si un événement de recombinaison se produit dans un intervalle entre marqueurs (de position connue) où le TIM n'est pas présent, alors $\phi(\mathbf{H}_\tau, \mathbf{H}_{\tau+1}) = 1$. Par ailleurs, $\phi(\mathbf{H}_\tau, \mathbf{H}_{\tau+1})$ est différent de 1 aussitôt qu'un événement de recombinaison se produit dans l'intervalle portant le TIM, c'est-à-dire dès que δ_p^l ou δ_p^r sont différents de 0. Ainsi, si le TIM est le $m^{\text{ième}}$ locus dans la séquence, $\phi(\mathbf{H}_\tau, \mathbf{H}_{\tau+1})$ peut aussi s'écrire :

$$\phi(\mathbf{H}_\tau, \mathbf{H}_{\tau+1}) = \begin{cases} \theta/\theta_0 & \text{si } Mu \\ \rho_p/\rho_{0p} & \text{si } Re(p), \text{ et que } p = m - 1 \text{ ou } p = m \\ 1 & \text{si } Re(p), \text{ et que } p \neq \{m - 1, m\} \\ 1 & \text{si } Co, \end{cases}$$

où ρ_{0p} est le paramètre ρ_p sous Θ_0 . La méthode proposée dans ce travail évalue la vraisemblance de r_T le long de la séquence et implique l'évaluation de Q_{Θ_0} pour $\Theta_0 = \{\{\theta_0, r_{T_1}\}, \{\theta_0, r_{T_2}\}, \dots, \{\theta_0, r_{T_{L-2}}\}\}$, c'est-à-dire l'utilisation de $L - 2$ ensembles de valeurs conductrices, une valeur conductrice pour chaque intervalle entre marqueurs. Notons que nous n'utiliserons qu'une valeur conductrice pour le paramètre θ (nous le supposons en fait connu dans les applications), mais plusieurs valeurs pourraient être utilisées en combinaison avec différentes valeurs de r_T . Pour la valeur conductrice r_{T_p} , nous prenons le milieu de l'intervalle entre marqueurs de position connue p ($1 \leq p \leq L - 2$), r_{T_p} est donc dans l'intervalle p . La vraisemblance dans l'intervalle p pour les autres valeurs que r_{T_p} est évaluée par la méthode d'échantillonnage d'importance décrite plus haut: ainsi, les graphes construits avec une valeur conductrice r_{T_p} sont utilisés pour évaluer la vraisemblance dans la région $[x_p, x_{p+1}]$ de la séquence.

Puisque les facteurs n_j et n_k apparaissent dans les probabilités de transition, nous devons les calculer à chaque étape. Cela peut être coûteux en temps. On peut alors changer la probabilité de transition d'une recombinaison: une transition est faite de \mathbf{H}_τ à $(\mathbf{H}_\tau + R_i^{jk}(p))$ avec probabilité $\rho_p/[(n+1)S'_{\mathbf{H}_\tau}]$, et alors, selon le type du $(\tau + 1)^{\text{ième}}$ événement, $f(\mathbf{H}_\tau, \mathbf{H}_{\tau+1})$ se définit comme:

$$f(\mathbf{H}_\tau, \mathbf{H}_{\tau+1}) = \begin{cases} S'_{\mathbf{H}_\tau}/D_{\mathbf{H}_\tau} & \text{si } Co \text{ ou } Mu, \\ (n_j + 1)(n_k + 1)S'_{\mathbf{H}_\tau}/D_{\mathbf{H}_\tau} & \text{si } Re. \end{cases} \quad (3.5)$$

où $S'_{\mathbf{H}_\tau}$ est alors définie comme suit:

$$S'_{\mathbf{H}_\tau} = n \sum_{\substack{i, \\ n_i > 1}} (n_i - 1) + 2n \sum_{\substack{i \neq j \\ \text{compatibles}}} (n_k + 1 - \delta_{ik} - \delta_{jk})$$

$$+ \sum_i \sum_{\substack{m \in A_i, s_m^i \neq s_m^j \\ \text{pour tout } k \neq i}} \theta(n_j + 1)/L + \sum_i \sum_{p=\gamma_i}^{\kappa_i} \rho_p / [(n + 1)].$$

Il existe en fait de nombreuses possibilités de choix pour les probabilités de transition, puisque la fonction $f(\mathbf{H}_\tau, \mathbf{H}_{\tau+1})$ peut être ajustée en conséquence.

Si on utilise cette simplification concernant les probabilités de transition des recombinaisons, la fonction f_{Θ_0} est alors définie par (3.5), et $\phi(\mathbf{H}_\tau, \mathbf{H}_{\tau+1})$ devient (nous le dénoterons alors ϕ'):

$$\phi'(\mathbf{H}_\tau, \mathbf{H}_{\tau+1}) = \begin{cases} \theta/\theta_0 & \text{si Mu} \\ (n_j + 1)(n_k + 1) \frac{r_p(1-\delta_p^l)(1-\delta_p^r) + r_l\delta_p^l + r_r\delta_p^r}{r_p(1-\delta_p^l)(1-\delta_p^r) + r_{0l}\delta_p^l + r_{0r}\delta_p^r} & \text{si Re} \\ 1 & \text{si Co.} \end{cases}$$

Nous n'utiliserons pas cette modification dans les simulations de la thèse, mais nous la présentons pour illustrer la flexibilité de la procédure.

Notons qu'en pratique, les résultats sont construits sur la distribution:

$$Q^*(\mathbf{H}_\tau) = L^{s_\tau} Q(\mathbf{H}_\tau),$$

où s_τ est le nombre de sites polymorphes à l'étape τ . Une équation de récurrence similaire à celle que nous avons déjà définie est alors utilisée:

$$\begin{aligned} Q^*(\mathbf{H}_\tau) &= \sum_{\substack{i, \\ n_i > 1}} \frac{S_{\mathbf{H}_\tau}}{D_{\mathbf{H}_\tau}} \frac{n(n_i - 1)}{S_{\mathbf{H}_\tau}} Q^*(\mathbf{H}_\tau + C_i) \\ &+ \sum_{\substack{i \neq j \\ \text{compatibles}}} \frac{S_{\mathbf{H}_\tau}}{D_{\mathbf{H}_\tau}} \frac{2n(n_k + 1 - \delta_{ik} - \delta_{jk})}{S_{\mathbf{H}_\tau}} Q^*(\mathbf{H}_\tau + C_{ij}^k) \\ &+ \sum_i \sum_{\substack{m \in A_i, s_m^i \neq s_m^j \\ \text{pour tout } k \neq i}} \frac{S_{\mathbf{H}_\tau}}{D_{\mathbf{H}_\tau}} \frac{\theta(n_j + 1)}{S_{\mathbf{H}_\tau}} Q^*(\mathbf{H}_\tau + M_i^j(m)) \\ &+ \sum_i \sum_{p=\gamma_i}^{\kappa_i} \frac{S_{\mathbf{H}_\tau}}{D_{\mathbf{H}_\tau}} \frac{\rho_p(n_j + 1)(n_k + 1)}{S_{\mathbf{H}_\tau}(n + 1)} Q^*(\mathbf{H}_\tau + R_i^{jk}(p)), \end{aligned}$$

car $s^{\tau+1} = s^{\tau} - 1$ si un événement de mutation vient de se produire à l'étape τ . La seule différence est qu'un événement de mutation M_i^j se fait avec probabilité $\theta(n_j + 1)/n$ au lieu de $\theta(n_j + 1)/(nL)$.

3.5. Un exemple illustratif

Afin d'illustrer la méthode, nous présentons ici un exemple. Les données sont constituées d'un échantillon de 88 cas, avec 5 marqueurs de position connue sur une séquence de 0.9 M (soit environ 900 kb). La figure 3.5.1 qui suit présente un résultat typique de ce que l'on va présenter dans la suite de cette thèse. Nous allons donc en décrire les caractéristiques.

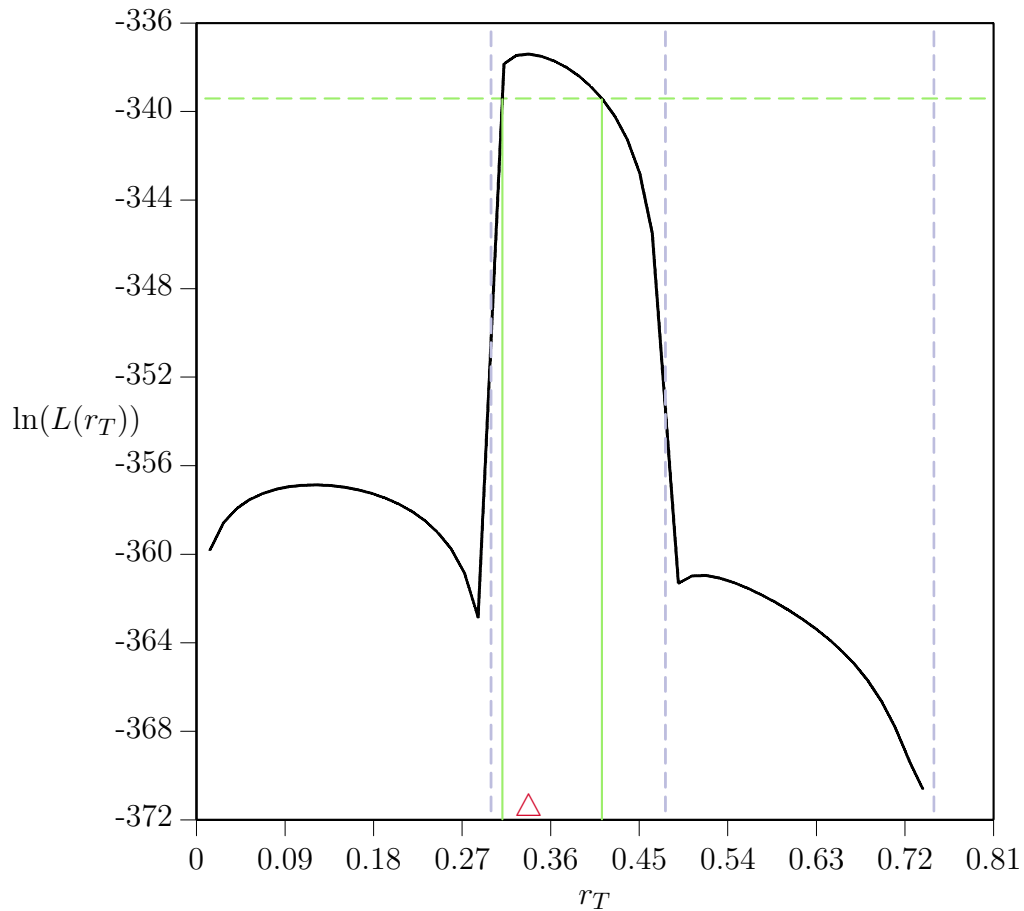


Figure 3.5.1. Exemple de résultat produit par la méthode.

Sur l'axe des abscisses, se trouve la coordonnée de la mutation cherchée (r_T); les unités sont des centiMorgans (cM). Le logarithme de la vraisemblance est représenté sur l'axe des ordonnées ($\ln(L(r_T))$). Bien que tous les marqueurs aient été utilisés dans le calcul de la vraisemblance, la vraisemblance n'a été calculée que sur les trois premiers intervalles entre marqueurs. Les lignes verticales en pointillé représentent la position de ces marqueurs, soit $x_1 = 0$, $x_2 = 0.3$, $x_3 = 0.48$ et $x_4 = 0.75$ ($x_5 = 0.9$ n'est pas représenté).

L'estimé du maximum de vraisemblance est $\hat{r}_T = 0.337714$, ce qui veut dire que la mutation cherchée serait située à environ 0.34 cM du premier marqueur. La position de l'estimé est indiquée par un triangle juste au-dessus de l'axe des abscisses. La ligne horizontale en pointillé à $y = -335.4$ représente deux unités en-dessous du maximum de vraisemblance, et l'on peut voir que l'intervalle de confiance approximatif à 95% en découlant est (0.31093; 0.41201) (la construction d'un intervalle de confiance est expliquée dans la prochaine section).

3.6. Intervalle de confiance

L'objectif principal de la méthodologie que l'on propose ici est d'obtenir une estimation de r_T . Mais il est important d'obtenir une estimation de la variation de notre estimé. Nous allons le faire en construisant un intervalle de confiance pour r_T . On sait que sous certaines conditions (Casella et Berger, 1990), l'estimateur du maximum de vraisemblance \hat{r}_T sera distribué selon une distribution normale d'espérance r_T , et que le rapport de vraisemblance suivant:

$$\lambda = -2 \ln \left[\frac{L(r_T)}{L(\hat{r}_T)} \right],$$

est distribué asymptotiquement selon une loi du χ^2 à un degré de liberté. Cela nous permet de construire des intervalles de confiance approximatifs. Ainsi, un intervalle de confiance approximatif à $100(1 - \alpha)\%$ pour r_T est donné par:

$$-2 \ln \left[\frac{L(r_T)}{L(\hat{r}_T)} \right] < q_{1,1-\alpha},$$

où $q_{1,1-\alpha}$ est le quantile $1-\alpha$ d'une densité d'une χ^2 à un degré de liberté. Notons que $L(\hat{r}_T)$ est toujours supérieur à $L(r_T)$, donc le rapport $L(r_T)/L(\hat{r}_T)$ est entre 0 et 1, et λ est toujours positif. L'intervalle de confiance est alors défini pour les valeurs de r_T qui satisfont:

$$\ln [L(r_T)] > \ln [L(\hat{r}_T)] - q_{1,1-\alpha}/2.$$

En pratique, on cherche souvent des intervalles de confiance à 95%, et alors, $q_{1,1-\alpha}/2 = 3.84/2 \approx 2$, ce qui permet d'obtenir l'intervalle de confiance en considérant simplement les valeurs de r_T pour lesquelles le logarithme de la vraisemblance est supérieur à deux unités en-dessous du maximum du logarithme de la vraisemblance.

La figure 3.6.1 (b) montre la vraisemblance relative, c'est-à-dire $L(r_T)/L(\hat{r}_T)$ et l'intervalle de confiance associé. Il est obtenu en considérant l'ensemble des r_T tel que:

$$\frac{L(r_T)}{L(\hat{r}_T)} > \exp\left(-\frac{q_{1,1-\alpha}}{2}\right),$$

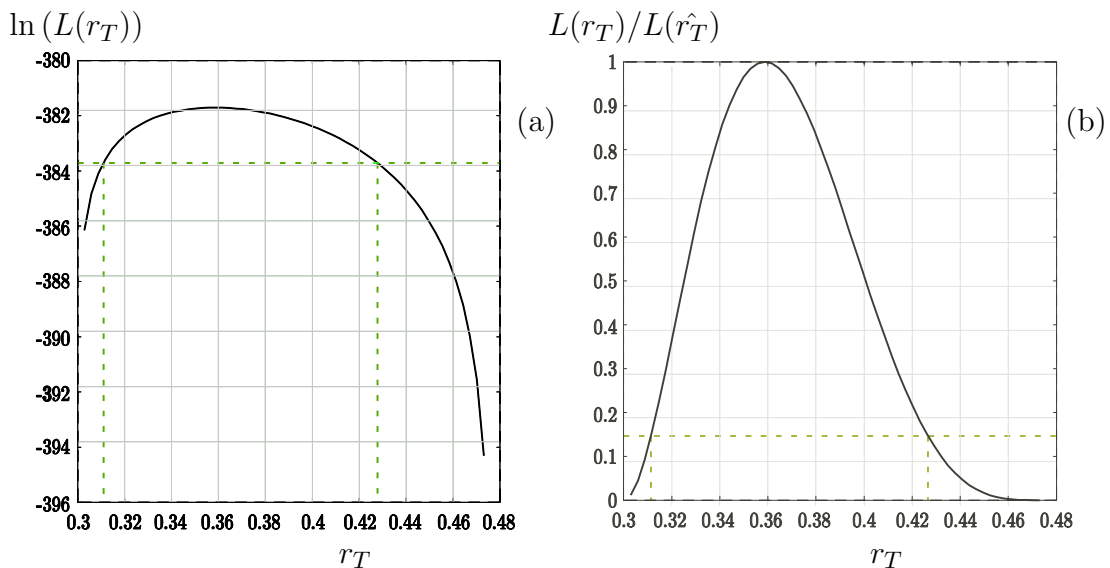


Figure 3.6.1. Exemple de calcul d'intervalle de confiance approximatif pour les données de l'exemple introduit dans la section précédente: (a) logarithme de la vraisemblance, (b) vraisemblance relative.

qui est l'équivalent de la procédure expliquée ci-dessus, mais pour la vraisemblance relative. Le logarithme de la vraisemblance et son intervalle de confiance associé sont illustrés dans la partie (a) de la figure. L'intervalle de confiance obtenu avec cet exemple est (0.31099; 0.42775).

Évidemment, il est difficile de connaître la qualité de ces intervalles de confiance dans ce contexte (Stephens, 2001). En effet, les données que nous utilisons ici ne sont pas indépendantes: par hypothèse, nous supposons que les séquences partagent une certaine histoire et un ancêtre commun. La théorie standard ne s'applique donc peut-être pas ici. Cependant, dans un contexte similaire au nôtre, Fearnhead et Donnelly (2001) montrent empiriquement des résultats qui supportent cette approche: ces intervalles approximatifs ont une bonne couverture. Cependant, compte tenu que l'on n'est pas dans les conditions régulières de la théorie nous permettant de construire ces intervalles, ils doivent être considérés avec précaution.

3.7. Description des ensembles de données

Trois ensembles de données ont été simulés selon un modèle neutre de Wright-Fisher (voir, par exemple Neuhauser (2001) ou Nordborg (2001)) par le programme `ms` (Hudson, 2002) en supposant une taille de population constante. Ce programme suppose une infinité de sites, et des mutations se produisent sur ces sites; seuls les sites polymorphiques sont alors considérés. Un échantillon de n séquences est généré avec un certain ρ , de sorte que l'on ait L marqueurs. Un de ces marqueurs est alors supposé être la mutation causant la maladie. On suppose donc que ce marqueur est à un emplacement inconnu, à une distance inconnue, et est en linkage complet avec le gène causant la maladie. Ce marqueur devient en fait le TIM, et le but de l'analyse est de trouver précisément son emplacement. Notons qu'une hypothèse doit être faite pour convertir ρ en distance génétique, c'est-à-dire en Morgans: nous supposons, comme on le fait habituellement que la taille effective de la population N est 10000 (Wall, 2003). De plus, les distances

généétiques sont converties en distances physiques tel que $1\text{cM} = 1 \text{ Mb}$. Donc, à partir de ρ , nous pouvons déduire $r = \rho/(4N)$, puis le nombre de kb (kilo bases) est obtenu en multipliant r par 1×10^5 . Ces trois exemples ont été choisis au hasard, mais on s'est assuré que la proportion des cas dans l'échantillon soit entre 5% et 95%, afin d'éliminer les échantillons qui n'auraient pas assez de cas et qui seraient difficiles à analyser; par ailleurs, notons que l'échantillonnage des séquences pour de telles études ne se fait jamais vraiment au hasard, puisque l'on échantillonne les individus en fonction de leur statut par rapport à une maladie ou un trait de caractère quelconque. Ces trois exemples présentent tous trois des difficultés différentes, comme nous allons le voir.

i	n_i	Séquence	TIM
1	15	■—■—■—□	■
2	3	■—■—□—■	■
3	6	■—■—■—□	□
4	2	■—■—□—■	□
5	1	■—□—□—■	■
6	2	■—□—■—□	□
7	1	□—■—■—□	■

Figure 3.7.1. Exemple A de données simulées: multiplicité de chaque séquence i (n_i) pour $i = 1, \dots, 7$, les séquences 3, 4 et 6 étant pour des cas, les autres pour les témoins. Le TIM occupe en fait la troisième position.

Le premier échantillon consiste en 30 séquences de 7 types différents, avec quatre marqueurs, en plus du TIM. On a 20 séquences de témoins et 10 séquences de cas. Les données sont illustrées à la figure 3.7.1. La position de la mutation est ici par construction au centre des autres marqueurs, dans le second intervalle. La longueur des séquences est $\rho = 60$. Nous référerons à ces données en parlant de l'exemple, ou des données, A.

i	n_i	Séquence	TIM
1	5	■ ■ — ■ ■	□
2	1	■ ■ — ■ □	■
3	1	■ ■ — ■ □	□
4	1	■ ■ — □ ■	□
5	1	□ ■ — ■ ■	□
6	1	□ □ — ■ □	■

Figure 3.7.2. Exemple B de données simulées: multiplicité de chaque séquence i (n_i) pour $i = 1, \dots, 6$, les séquences 1, 3, 4 et 5 étant pour des cas, les autres pour les témoins. Le TIM occupe en fait la troisième position.

Le second exemple, que nous noterons B, est constitué de 10 séquences de quatre marqueurs en plus du TIM, avec 6 types différents de séquences, tel que $\rho = 5$, ce qui donne $r = 0.0125\text{cM}$. Les trois distances entre marqueurs sont $r_1 = 0.0007382\text{cM}$, $r_2 = 0.0096382\text{cM}$ et $r_3 = 0.0021236\text{cM}$. La mutation cherchée est par construction dans le second intervalle, à une distance approximative de 0.007cM du premier marqueur ($r_T = 0.0069008993$).

Le troisième exemple, que nous noterons C, est constitué de 100 séquences de deux marqueurs encadrant le TIM, tel que $\rho = 20$. Alors, $r = 0.0005$, et la mutation est à la distance $r_T = 0.0096997351 \approx 0.0097$.

i	n_i	Séquence	TIM
1	78	■ — ■	□
2	9	■ — ■	■
3	6	■ — □	■
4	6	□ — ■	□
5	1	■ — □	□

Figure 3.7.3. Exemple C de données simulées: multiplicité de chaque séquence i (n_i) pour $i = 1, \dots, 5$, les séquences 1, 4 et 5 étant pour les cas, les autres pour les témoins. Le TIM occupe en fait la seconde position.

Dans l'exemple A, la proportion des cas est faible (30% de l'échantillon), et la séquence est assez grande (environ 150 kb). L'exemple B présente 80% de cas, mais pour un effectif total très faible, seulement 10 haplotypes. Le TIM se trouve dans le second intervalle, pas très loin du centre de la séquence, donc de la valeur

conductrice. La séquence est ici plus petite, de l'ordre de 12.5 kb. Le troisième exemple, C, se constitue de 100 séquences, dont 85% de cas. La séquence est d'environ 20 kb, et le TIM peut être difficile à trouver car très proche du début de la séquence.

Nous utiliserons également un exemple réel (que nous noterons D), celui de l'épilepsie myoclonique progressive. Il s'agit de données publiées dans Virtaneva *et al.* (1996). Nous avons 88 haplotypes cas et aucun témoin. Ces données sont originellement des microsatellites, que nous avons convertis en marqueurs binaires, en utilisant la séquence la plus fréquente dans l'échantillon comme représentative de l'état ancestral. Chaque haplotype contient cinq marqueurs autres que le TIM couvrant une région de 895 kb. Le gène de la maladie est maintenant connu, et a été cloné (Pennachio *et al.*, 1996); il est situé à 30kb après le second marqueur, c'est-à-dire que $r_T = 0.33cM$. Les distances entre les marqueurs de position connue sont, d'après Virtaneva *et al.* (1996), 300kb, 176kb, 276kb et 143kb. Nous avons assumé, comme on le fait habituellement, que $1cM \approx 1Mb$. La valeur de ρ est ici de 358, et $r = 0.00895$. L'avantage de travailler sur ces données est que d'autres méthodes d'analyse les ont déjà utilisées, et qu'il est tout à fait intéressant de comparer nos résultats avec d'autres méthodes.

i	n_i	Séquence	TIM
1	65	■—■—■—■—■	□
2	10	■—■—■—■—□	□
3	4	□—■—■—■—■	□
4	1	■—■—■—□—□	□
5	1	□—■—■—□—□	□
6	1	■—■—■—□—■	□
7	2	■—■—□—□—□	□
8	2	■—□—■—■—■	□
9	1	□—□—■—■—■	□
10	1	□—■—□—■—□	□

Figure 3.7.4. Exemple D de l'épilepsie myoclonique progressive. Multiplicité de chaque séquence i (n_i) pour $i = 1, \dots, 10$, toutes les séquences sont des cas. Le TIM occupe en fait la troisième position.

Enfin, nous utiliserons un autre exemple, noté E, qui correspond à des données de patients atteints de fibrose kystique. Ces données sont constituées de 94 haplotypes cas et de 92 témoins. Pour chacun d’eux, nous disposons de 23 marqueurs microsatellites. Nous décrirons ces données plus en détail[†] quand nous aborderons les marqueurs microsatellites (chapitre 5); nous utiliserons pour l’instant cet exemple seulement dans le but d’obtenir des ordres de grandeur de nombre d’événements impliqués dans les calculs (dans le chapitre 4 par exemple, où nous comparons plusieurs algorithmes).

3.8. Détails de la méthode

Nous allons étudier dans cette section le fonctionnement détaillé de la méthode. Nous avons modifié le programme pour garder certaines informations, comme \hat{r}_T et $L(\hat{r}_T)$ à chacune des itérations du processus, ceci afin de suivre les estimations dans le temps. Pour observer le comportement des estimations, plusieurs millions d’itérations sont nécessaires, et nous ne pouvons stocker tant d’informations en mémoire. Nous avons donc gardé en mémoire les informations qui nous intéressaient seulement quand on avait un changement dans l’estimation de r_T . Notons que cette version du programme est évidemment beaucoup plus longue, puisqu’à chaque itération, nous devons calculer \hat{r}_T (chercher la valeur maximum d’un vecteur) et le comparer à la valeur qu’il avait à l’itération précédente. Les exemples qui suivent illustrent le fonctionnement de la méthode proposée.

Dans ce qui suit, nous présenterons beaucoup de résultats basés sur un certain nombre d’itérations. Nous noterons par “m” un nombre d’itérations en milliers, et par “M” un nombre d’itérations en millions. Aussi, il faut comprendre que si l’on mentionne K millions d’itérations, cela signifie que KM d’itérations sont effectuées dans chacun des intervalles. Si par exemple les données sont constituées de $L - 1$ marqueurs de position connue, il faut chercher r_T dans $L - 2$ intervalles,

[†] cf. section 5.3

et donc $K \times (L - 2)M$ de graphes (itérations) sont en réalité faits pour évaluer la vraisemblance.

La figure 3.8.1 montre le suivi des estimations pour 100M d'itérations pour les données de l'exemple B. La partie (a) montre toute la suite des estimations de la première à la 100 millionième, tandis que la partie (b) de la figure ne montre les estimations qu'à partir de 1M, ceci pour mieux observer les variations de la vraisemblance. L'échelle de gauche du graphique représente l'échelle de \hat{r}_T (ligne pleine) tandis que l'échelle de droite représente l'échelle de $\ln(L(\hat{r}_T))$ (ligne en pointillé).

On peut voir sur la partie (a) que la vraisemblance se stabilise assez rapidement: après un ou deux millions d'itérations, elle n'augmente plus. L'estimation de r_T ne se stabilise autour de la valeur 0.007 qu'après 17 millions d'itérations. La partie (b) nous montre plus clairement la variation de la vraisemblance. On peut

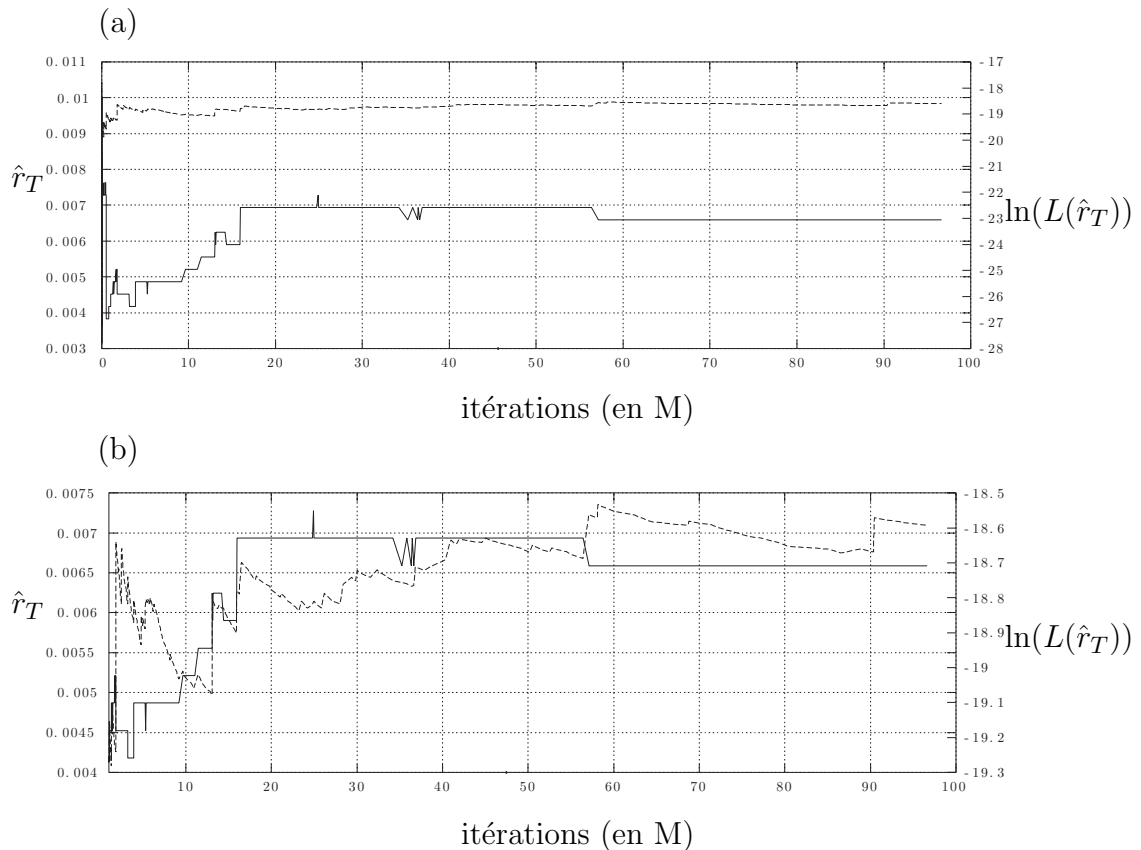


Figure 3.8.1. Résultats d'un premier ensemble de simulations pour les données de l'exemple B: (a) itérations de 1 à 100M, (b) itérations de 1M à 100M.

voir que celle-ci augmente brusquement quand un graphe ayant un poids élevé est trouvé, puis diminue jusqu'à ce qu'un autre graphe de poids élevé la fasse de nouveau augmenter.

La figure 3.8.2 nous montre, pour un autre ensemble de simulations, un schéma similaire au précédent. La vraisemblance converge rapidement, bien que \hat{r}_T ne trouve sa bonne valeur qu'après 20M d'itérations. La partie (b) nous montre clairement la décroissance de la vraisemblance en fonction du nombre d'itérations du processus.

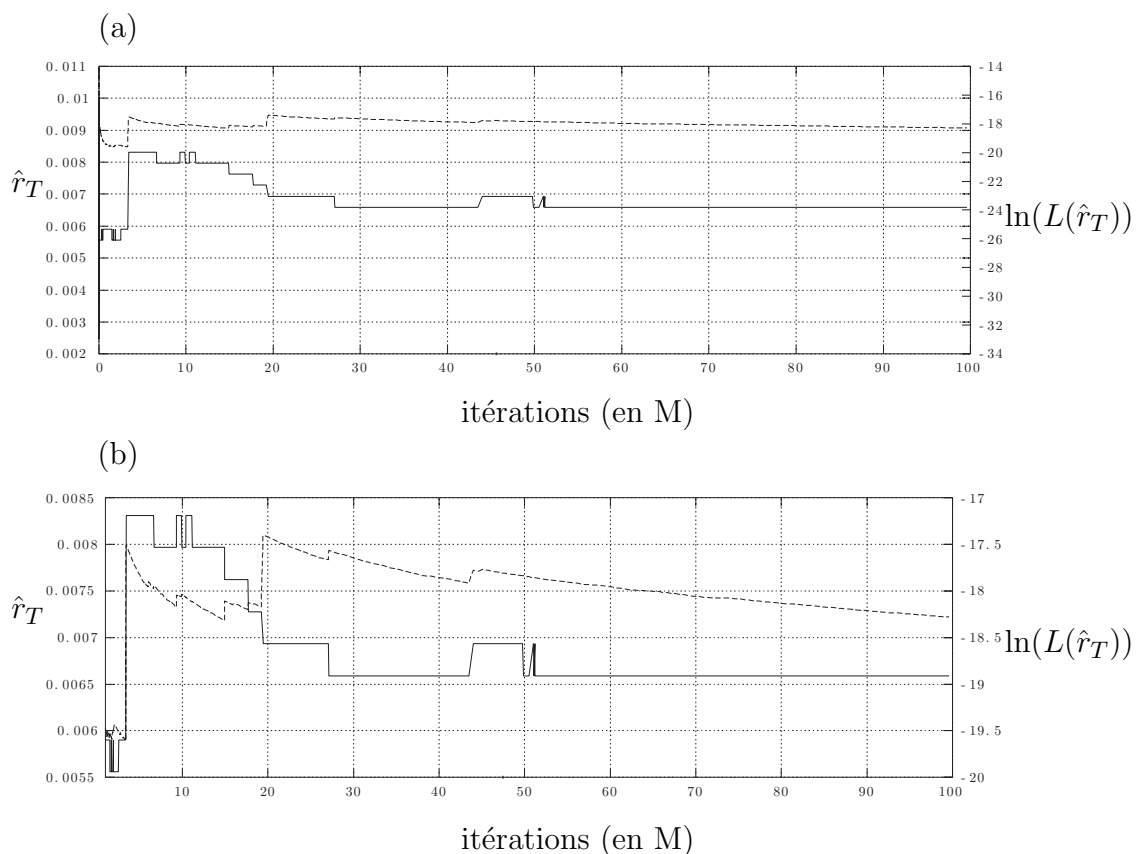


Figure 3.8.2. Résultats d'un second ensemble de simulations pour les données de l'exemple B: (a) itérations de 1 à 100M, (b) itérations de 1M à 100M.

Les figures 3.8.3 et 3.8.4 nous montrent deux exemples des suivis de \hat{r}_T et $L(\hat{r}_T)$ pour les données de l'exemple C, le premier pour un ensemble de 50M d'itérations et le second pour un autre ensemble de 100M. On voit que la vraisemblance ne semble pas avoir convergée au bout des 50M d'itérations, car nous ne pouvons observer de stabilisation (fig. 3.8.3). Par contre, la vraisemblance se sta-

bilise très rapidement pour le cas des 100M d'itérations (fig. 3.8.4). L'estimation de r_T ne semble pas très stable, mais semble converger vers une valeur proche du début de la séquence.

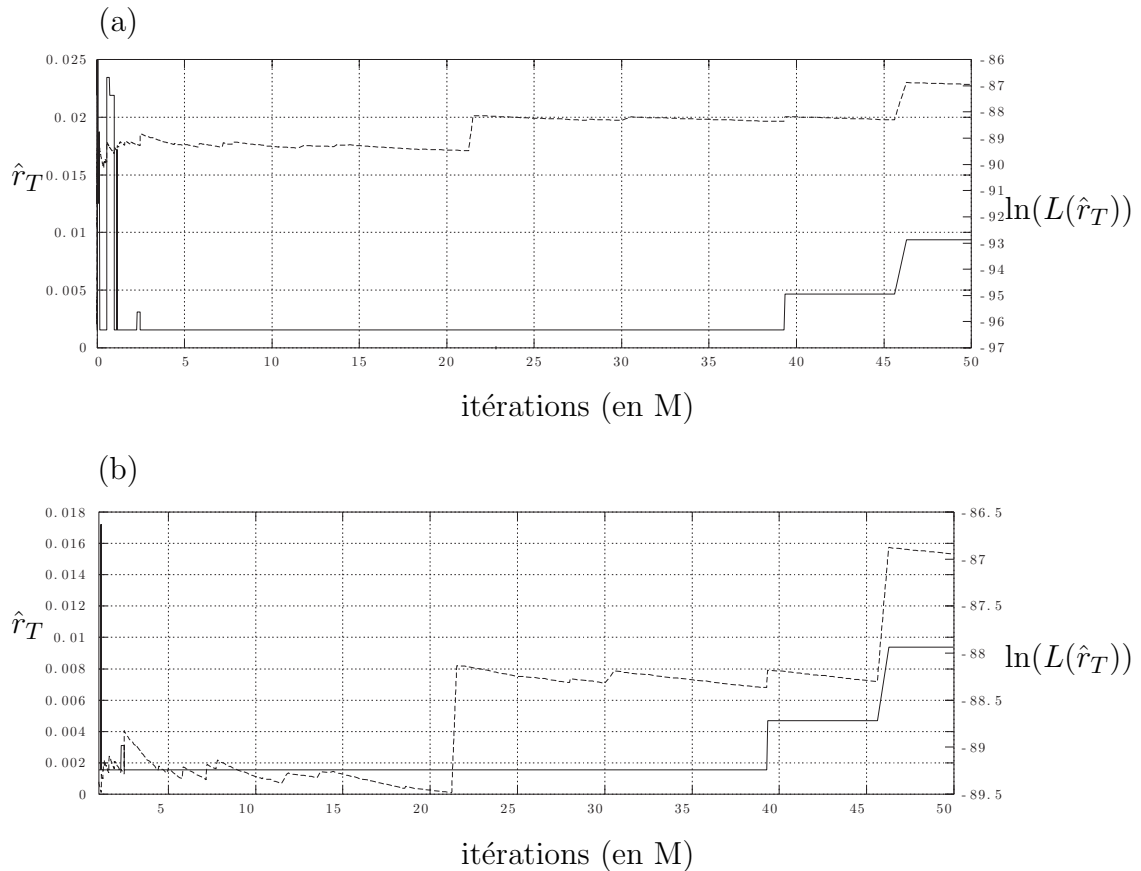


Figure 3.8.3. Résultats d'un premier ensemble de simulations pour les données de l'exemple C: (a) itérations de 1 à 50M, (b) itérations de 1M à 50M.

Ces illustrations nous montrent bien que le nombre de graphes qui contribuent réellement à la vraisemblance est faible: la vraisemblance augmente substantiellement chaque fois que l'on trouve de tels graphes. Il faut cependant interpréter les faits avec précaution. On pourrait d'abord penser que la distribution proposée qui construit les graphes n'est pas bonne, mais en essayant de reconstruire des généalogies dans le passé, nous cherchons des événements rares: les recombinaisons qui "expliquent" les données observées sont difficiles à trouver, et de longues recherches sont nécessaires.

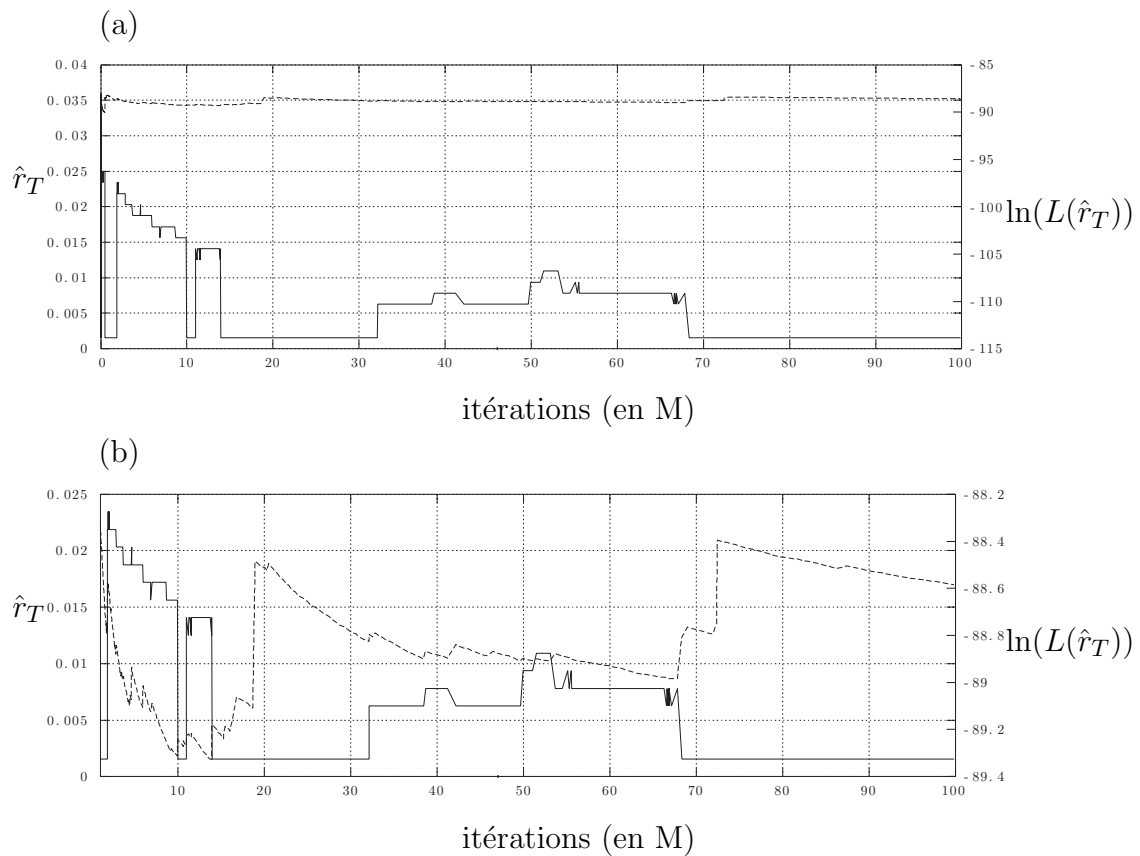


Figure 3.8.4. Résultats d'un second ensemble de simulations pour les données de l'exemple C: (a) itérations de 1 à 100M, (b) itérations de 1M à 100M.

3.9. Test de la méthode

Compte tenu du temps d'exécution assez grand, il est difficile de faire un ensemble de tests par simulation pour voir si la méthode évalue de façon correcte l'estimateur cherché et d'évaluer sa puissance, choses que l'on fait habituellement quand on fait le développement d'une nouvelle méthode. Notons par ailleurs que ce problème se pose pour la plupart des méthodes de cartographie qui existent. Comme plusieurs auteurs, nous nous contenterons de tester la méthode par des exemples simples, construits à partir de données simulées, et sur des exemples basés sur des données réelles bien connues. Nous pourrions alors essayer de comparer nos estimations à celles d'autres méthodes. Notons enfin que dans les résultats de simulations, le terme "intervalle" réfère systématiquement à un intervalle en-

tre marqueurs de position connu. De plus, nous avons omis volontairement la légende des axes sur tous les graphiques montrant les profils de vraisemblance, ceci afin d'alléger la présentation. L'axe des ordonnées désigne systématiquement $\ln(L(\hat{r}_T))$, et l'axe des abscisses désigne r_T .

La figure 3.9.1 illustre des résultats de simulations basées sur 1M d'itérations (a), 5M d'itérations (b), 10M d'itérations (c) et 20M d'itérations (d) pour l'exemple A. Pour chacun de ces cas, le graphique représente cinq courbes de vraisemblance obtenues à partir d'itérations indépendantes (en pâle), et la courbe cumulée, basée donc sur 25M d'itérations pour (a), ou par exemple sur 100M d'itérations pour

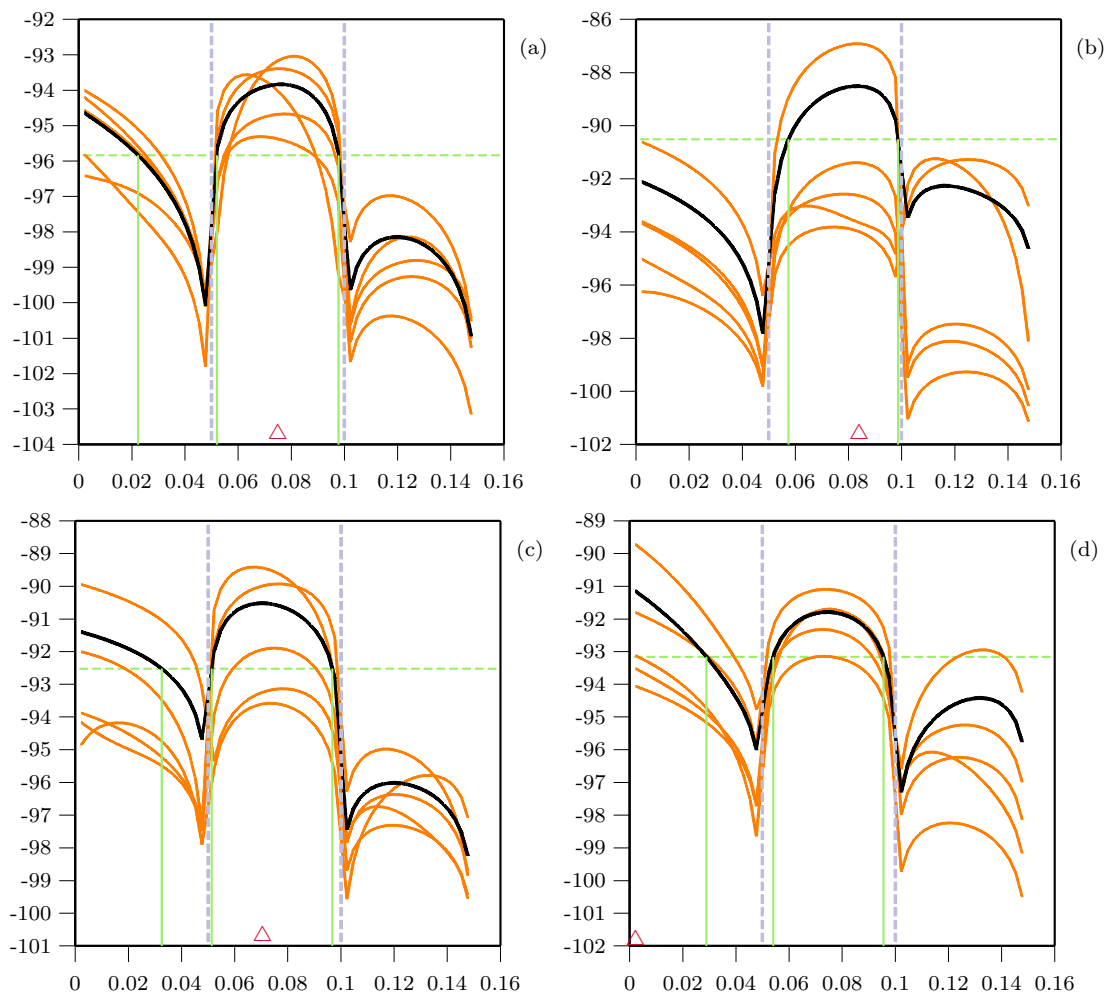


Figure 3.9.1. Courbes de vraisemblance pour l'exemple A: (a) 1M d'itérations chaque, (b) 5M d'itérations chaque, (c) 10M d'itérations chaque et (d) 20M d'itérations chaque. Les courbes en gras sont pour les itérations cumulées.

(d). Précisons encore que quand nous parlons d'estimations obtenues sur 10M d'itérations (par exemple), 10M d'itérations sont effectuées dans chacun des $L-2$ intervalles entre marqueurs, donc comme nous avons ici trois intervalles entre marqueurs, 30M de graphes sont construits pour évaluer le profil de vraisemblance. Comme on peut le voir sur la figure 3.9.1, nous obtenons des profils de vraisemblance très similaires, quel que soit le nombre d'itérations utilisé. L'estimé de r_T est bien dans le second intervalle comme il se doit, sauf pour le cas (d) utilisant 20M d'itérations, où l'estimateur de r_T se place dans le premier intervalle. On remarque cependant que dans ce cas, l'intervalle de confiance inclut le second

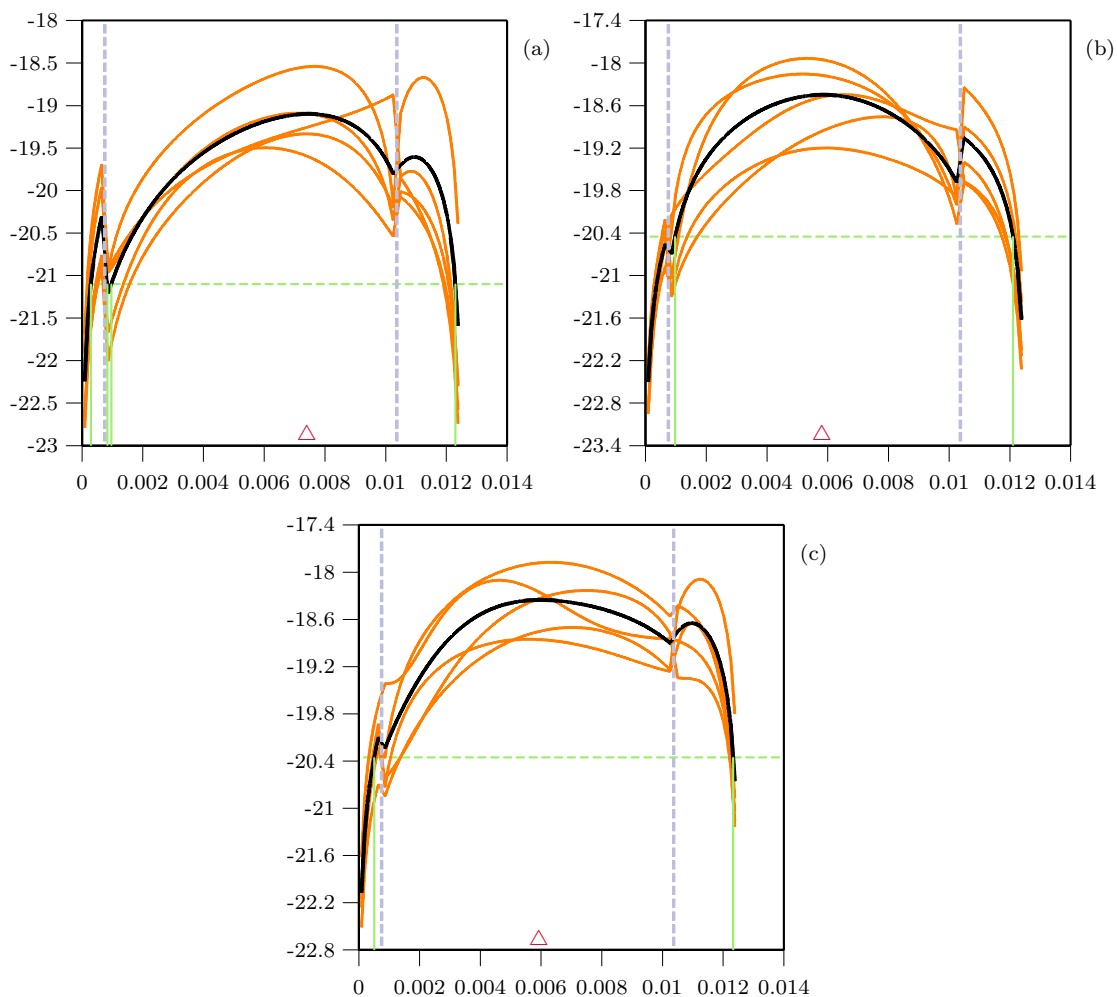


Figure 3.9.2. Courbes de vraisemblance pour l'exemple B: (a) 1M d'itérations chaque, (b) 5M d'itérations chaque et (c) 20M d'itérations chaque. Les courbes en gras sont les courbes cumulées.

intervalle. Néanmoins, on constate une variabilité des différents profils de vraisemblance. Notons par ailleurs que cette variabilité ne semble pas diminuer avec le nombre d'itérations.

Les résultats correspondants pour l'exemple B sont présentés à la figure 3.9.2, qui montre des courbes de vraisemblance basées sur 1M, 5M et 20M d'itérations chaque fois. Les estimés obtenus pour la position du TIM varient entre 0.006 et 0.007, alors la valeur réelle de r_T est ici de 0.0069. Les estimations sont donc globalement bonnes, bien qu'ici aussi des variations peuvent être observées entre les différents profils de vraisemblance.

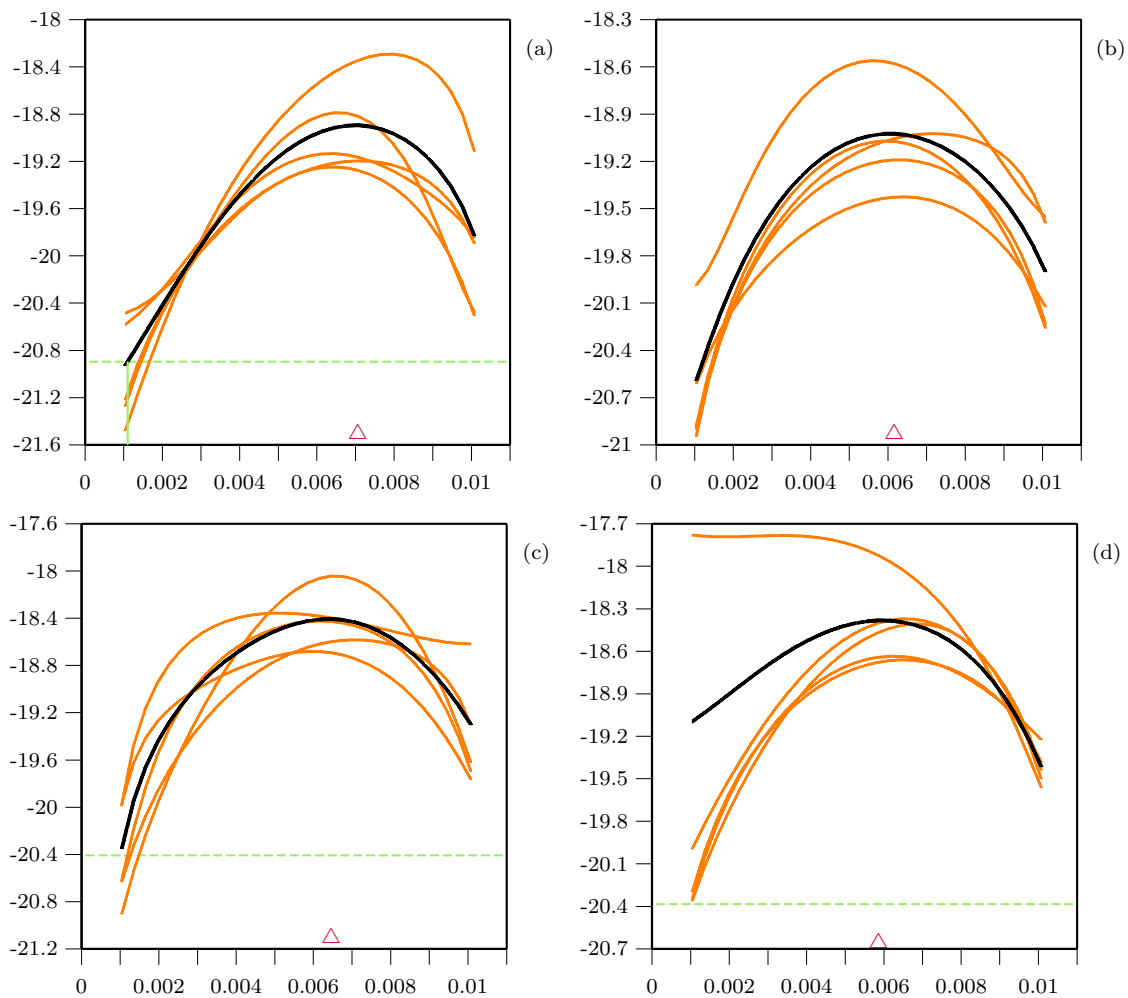


Figure 3.9.3. Courbes de vraisemblance pour le second intervalle de l'exemple B: (a) 1M d'itérations chaque, (b) 5M d'itérations chaque, (c) 20M d'itérations chaque et (d) 100M d'itérations chaque.

Maintenant que nous savons que la mutation est localisée dans le second intervalle, nous pouvons effectuer d'autres analyses complémentaires en se concentrant sur cet intervalle. La figure 3.9.3 évalue la courbe de vraisemblance sur cet intervalle, pour 1M, 5M, 20M et 100M d'itérations. Notons que bien que la vraisemblance soit évaluée seulement pour le second intervalle, les quatre marqueurs sont utilisés pour construire les graphes. Ici encore, les profils sont assez similaires, et il est intéressant de voir que la variation des profils de vraisemblance ne diminue pas avec l'augmentation du nombre d'itérations. La figure 3.9.3 (d) montre qu'une des cinq courbes semble "aberrante"; elle est très différente des autres, tant par sa hauteur que par l'estimé de r_T qu'elle donne. Notons par ailleurs que les courbes de vraisemblance sont similaires quel que soit le nombre d'itérations effectuées.

Les résultats pour l'exemple C sont présentés à la figure 3.9.4, pour 1M, 5M, 10M et 100M d'itérations. Dans cet exemple, la mutation que nous cherchons est à la position 0.009, c'est-à-dire très proche du début de la séquence. Quel que soit le nombre d'itérations utilisé, tous les profils nous donnent un estimé assez proche de la vraie valeur. L'estimation avec 1M d'itérations est presque parfaite, alors que l'estimation avec 5M d'itérations donne un estimé au tout début de la séquence; on a donc un estimé du taux de recombinaison entre le premier marqueur et la mutation qui est 0, ce qui veut dire que la séquence serait en linkage total avec le premier marqueur. Enfin, les estimations basées sur 10M et 100M d'itérations donnent des estimés légèrement trop élevés. Bien qu'il faille être prudent dans l'interprétation des intervalles de confiance, ils sont tous très semblables, et, comme on peut le voir, pas très informatifs puisqu'ils couvrent une grande partie de la séquence.

Les trois exemples que nous venons d'étudier montrent que la méthode est apte à localiser la mutation, et ce dans des situations assez différentes, en ce qui concerne le nombre de séquences de l'échantillon, la longueur des séquences, et la position de la mutation par rapport aux marqueurs disponibles. Nous voyons

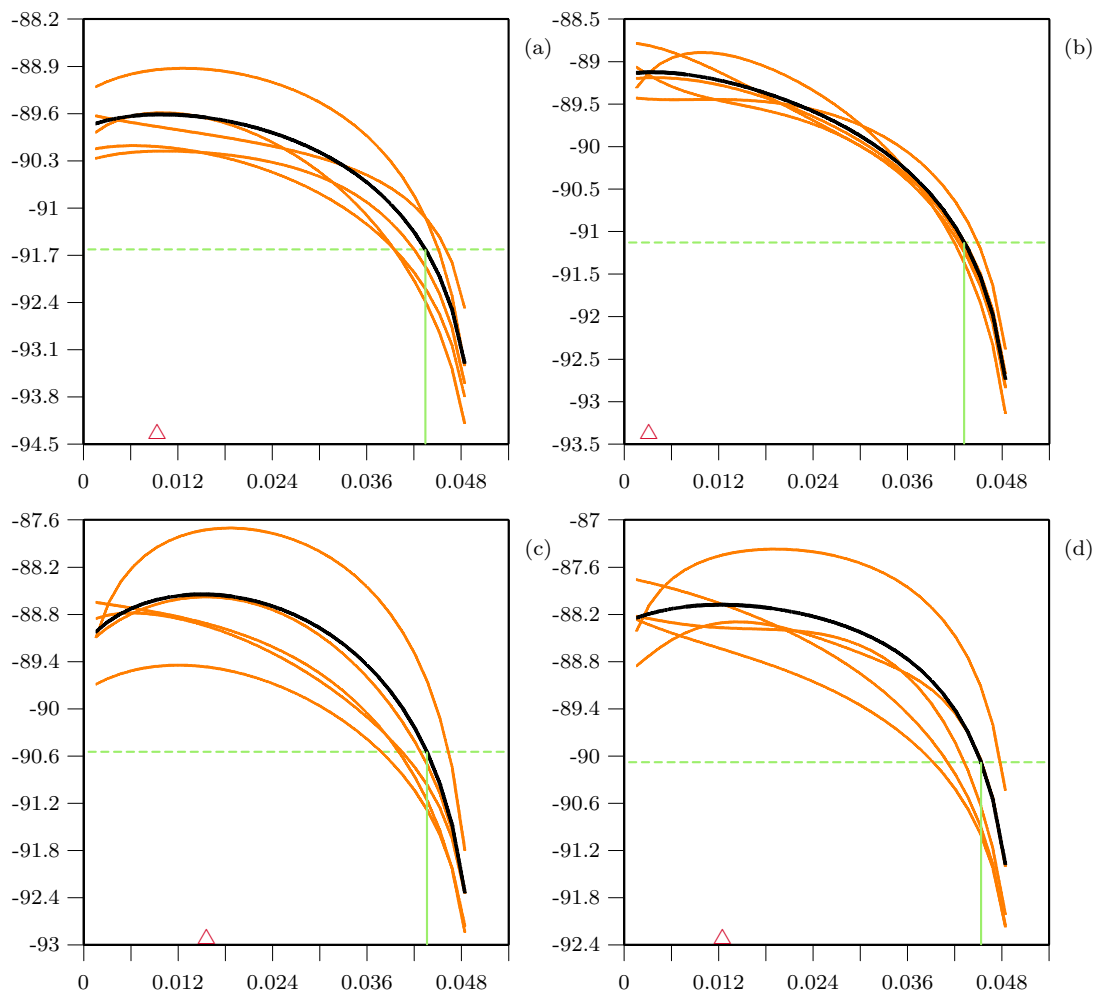


Figure 3.9.4. Courbes de vraisemblance pour l'exemple C: (a) 1M d'itérations chaque, (b) 5M d'itérations chaque, (c) 10M d'itérations chaque et (d) 100M d'itérations chaque. Les courbes en gras sont pour les itérations cumulées.

que les profils de vraisemblance montrent une variation non négligeable, qui ne semble pas diminuer avec le nombre d'itérations.

Notons que les temps de calcul sont assez grands; par exemple, cela nécessite 6 jours 12 heures de calcul pour évaluer les 100M de graphes (cf. fig. 3.9.1(d)) de l'exemple A, 2 jours 15 heures de calcul les 500M d'itérations de l'exemple B (cf. fig. 3.9.3(d)), et 4 jours 12 heures pour les 500M de graphes de l'exemple C (cf. fig. 3.9.4(d))[†]. Évidemment, plusieurs machines sont utilisées en même temps,

[†] Ces calculs sont effectués sur un processeur Pentium III, 1GHz

ce qui rend les estimations possibles en un temps raisonnable. En principe, plus le nombre d'itérations utilisé pour évaluer la vraisemblance est grand, moins la variation des courbes de vraisemblance devrait être grande. La vitesse de convergence de la méthode est un des facteurs importants d'une telle méthode. Nous allons décrire cette convergence. Notons que pour vérifier la convergence de telles méthodes, la meilleure solution consiste à reproduire plusieurs fois l'évaluation d'une même vraisemblance de façon indépendante, et de vérifier si les résultats concordent. Nous utilisons les exemples B et C pour illustrer cette vitesse de convergence.

La figure 3.9.5 présente ces résultats de simulations pour l'exemple C. Six simulations ont été effectuées, avec chacune un nombre d'itérations différent. Les figures (a)–(f) présentent respectivement les résultats pour 50m, 100m, 500m, 1M, 2M et 5M d'itérations. Pour chacune, dix simulations indépendantes avec le même nombre d'itérations sont effectuées. La courbe de vraisemblance combinée des dix analyses est indiquée par une courbe en gras. Le temps total nécessaire pour les simulations a–f sont respectivement, 5m, 10m, 51m, 1h40m, 4h19m et 8h22m.

Les quatre premières simulations ((a)–(d)) montrent un estimé très proche de la vraie valeur de r_T , tandis que les deux dernières, bien qu'étant basées sur plus d'itérations, présentent un estimé un peu moins bon. Plus le nombre d'itérations augmente, plus la variation semble diminuer, mais le phénomène est très lent. Bien que beaucoup d'itérations soient utilisées ((e) et (f)), certaines courbes de vraisemblance montrent un comportement assez éloigné de la courbe cumulée.

Un même schéma de simulation a été effectué pour l'exemple B (cf. fig. 3.9.6). Les remarques et conclusions sont semblables à l'exemple précédent. On voit que l'estimé de r_T est très stable, et toujours très proche de sa vraie valeur, soit environ 0.007. Les temps de calcul totaux nécessaires pour les simulations a–f sont respectivement 4m, 8m, 40m, 1h21m, 2h40m et 20h16m. La simulation qui semble la plus stable est celle effectuée à partir de 5M d'itérations (e).

La variation observée des courbes de vraisemblance sur des simulations indépendantes diminue avec le nombre d'itérations, mais très lentement. Par contre, il est positif de constater que peu d'itérations suffisent pour avoir une

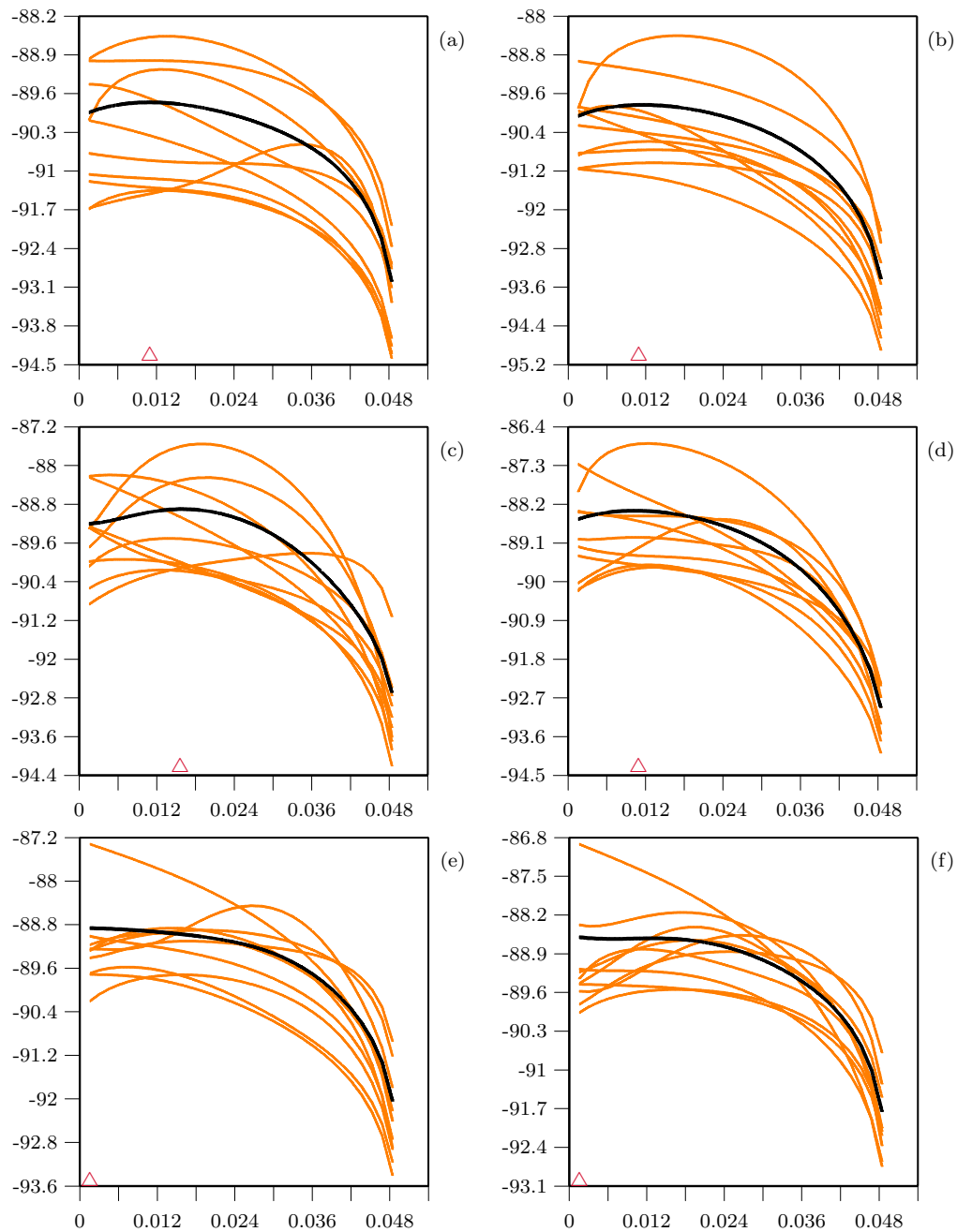


Figure 3.9.5. Courbes de vraisemblance pour l'exemple C: (a) 50m d'itérations chaque, (b) 100m d'itérations chaque, (c) 500m d'itérations chaque, (d) 1M d'itérations chaque, (e) 2M d'itérations chaque et (f) 5M d'itérations chaque.

image globale adéquate de la vraisemblance, du moment que de multiples analyses indépendantes sont effectuées. Ceci est important, car le temps de calcul devient réellement un enjeu sur des données réelles contenant beaucoup de marqueurs

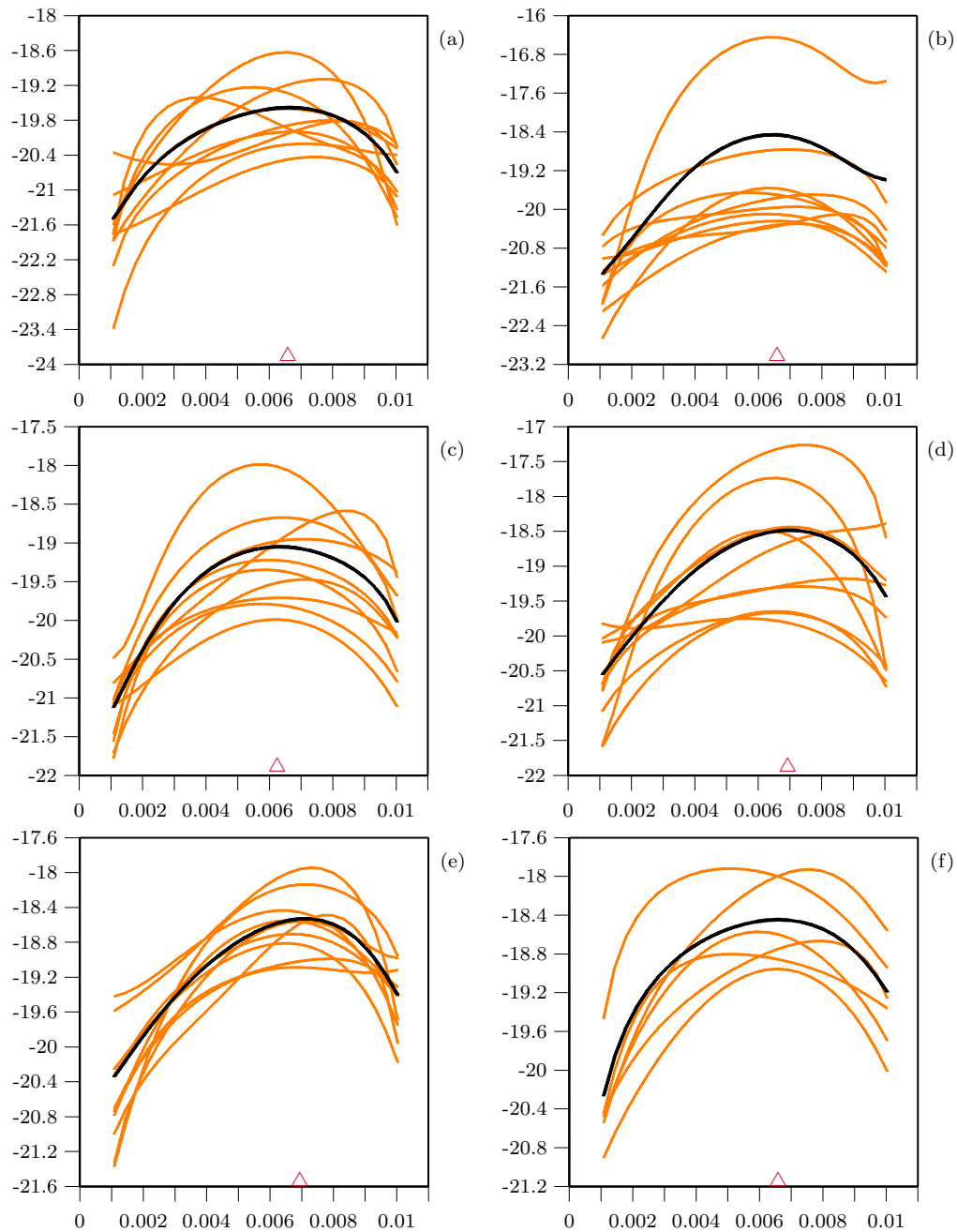


Figure 3.9.6. Courbes de vraisemblance pour l'exemple B: (a) 50m d'itérations chaque, (b) 100m d'itérations chaque, (c) 500m d'itérations chaque, (d) 1M d'itérations chaque, (e) 5M d'itérations chaque et (f) 25M d'itérations chaque.

et de séquences. Évidemment, il faut rester prudent, car nous ne pouvons pas extrapoler des observations faites sur deux exemples seulement.

Pour finir cette section, nous présentons des résultats de simulations pour l'exemple D. Rappelons que la mutation cherchée est ici dans le second intervalle, à la position 0.33 cM. La figure 3.9.7 montre des courbes de vraisemblance basées sur 100m d'itérations (a), 250m d'itérations (b), 1M d'itérations (c) et 50M d'itérations (d). Comme on le voit, il y a beaucoup de variations dans les courbes de vraisemblance. Remarquons par contre que les profils globaux de (a) (b) et (c) sont similaires. La simulation de 50M d'itérations montre un profil différent, mais a été effectuée avec un taux de mutation plus élevé. Chaque courbe de 50M d'itérations a demandé 36 jours de calcul, elle a été calculé alors que nous n'avions

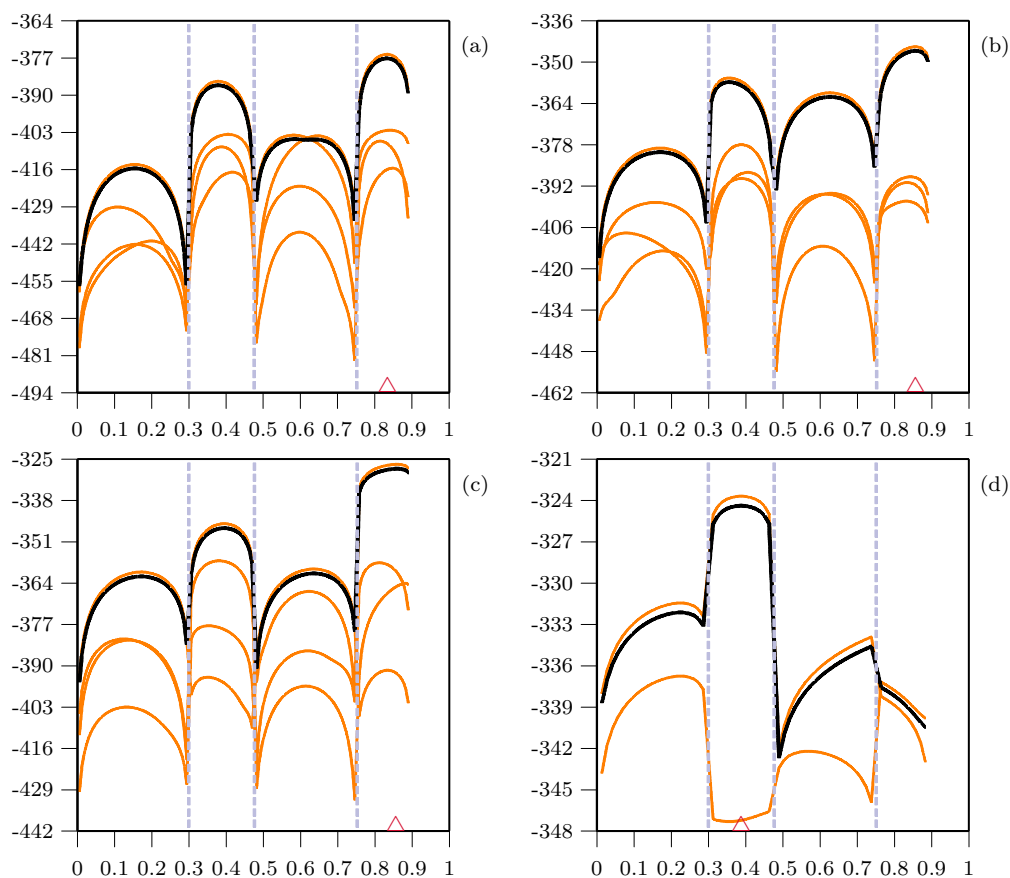


Figure 3.9.7. Courbes de vraisemblance pour l'exemple D: (a) 100m d'itérations chaque, (b) 250m d'itérations chaque, (c) 1M d'itérations chaque, (d) 50M d'itérations chaque; (a), (b) et (c) $u = 6 \times 10^{-5}$, (d) $u = 8 \times 10^{-3}$.

pas encore optimisé notre programme. Plusieurs raisons peuvent expliquer la grande variation des courbes de vraisemblances que l'on observe avec les données de cet exemple: la séquence est très longue, et nous avons peu de marqueurs, donc peu d'informations. La distance entre chaque marqueur étant relativement grande, il y a peu de dépendance entre l'histoire de chaque locus. Griffiths (1981), en supposant un modèle à deux loci a et b , a montré que la corrélation entre les temps t_a et t_b jusqu'aux MRCAs aux deux loci est

$$\text{Cor}(t_a, t_b) = \frac{\rho + 18}{\rho^2 + 13\rho + 18}.$$

Comme ici, ρ est 358, on obtient que la corrélation entre les temps aux ancêtres communs entre le premier et dernier locus est 0.003, ce qui est très faible. Il est par conséquent difficile de reconstruire l'histoire commune des séquences à l'aide du graphe de recombinaison ancestral. De plus, comme nous l'avons déjà dit, il s'agit ici de marqueurs microsatellites recodés en binaire pour pouvoir utiliser un modèle à mutation non récurrente, ce qui n'est probablement pas une description adéquate de l'histoire.

3.10. Valeur conductrice

On a vu que l'estimation par vraisemblance de r_T dans l'intervalle $[x_p, x_{p+1}]$ était générée avec la valeur conductrice r_{T_p} qui est le milieu de l'intervalle entre marqueurs p ($p = 1, \dots, L - 2$), et que la vraisemblance pour les autres valeurs de r_T dans le même intervalle est estimée en introduisant la fonction $h_{\Theta\Theta_0}(\mathbf{H}_\tau \mathbf{H}_{\tau+1})$ dans $Q_\Theta(\mathbf{H}_\tau)$ tel que décrit par l'équation (3.4). L'inconvénient de ce type d'approche est que l'on risque d'obtenir une mauvaise estimation de la vraisemblance si la vraie valeur de r_T s'éloigne trop de la valeur conductrice. Notons que pour contrer ce problème, Fearnhead et Donnelly (2001) font de l'échantillonnage par pont (*bridge sampling*): plusieurs valeurs conductrices sont utilisées et la vraisemblance finale est un mélange obtenue à partir des différentes vraisemblances de chacune des valeurs conductrices.

Afin de voir l'effet des valeurs conductrices sur les estimations produites par la méthode, nous avons modifié le programme pour qu'il utilise plusieurs valeurs conductrices. La vraisemblance est ainsi estimée de façon indépendante pour chaque point, c'est-à-dire qu'aucune fonction ne déduit la vraisemblance des autres points en fonction de la valeur d'un seul point, comme on le fait habituellement. Les figures 3.10.1, 3.10.2 et 3.10.3 montrent chacune deux estimations indépendantes de la vraisemblance à partir de 1M, 2M et 5M d'itérations respectivement, en utilisant 120 valeurs conductrices sur les données de l'exemple C.

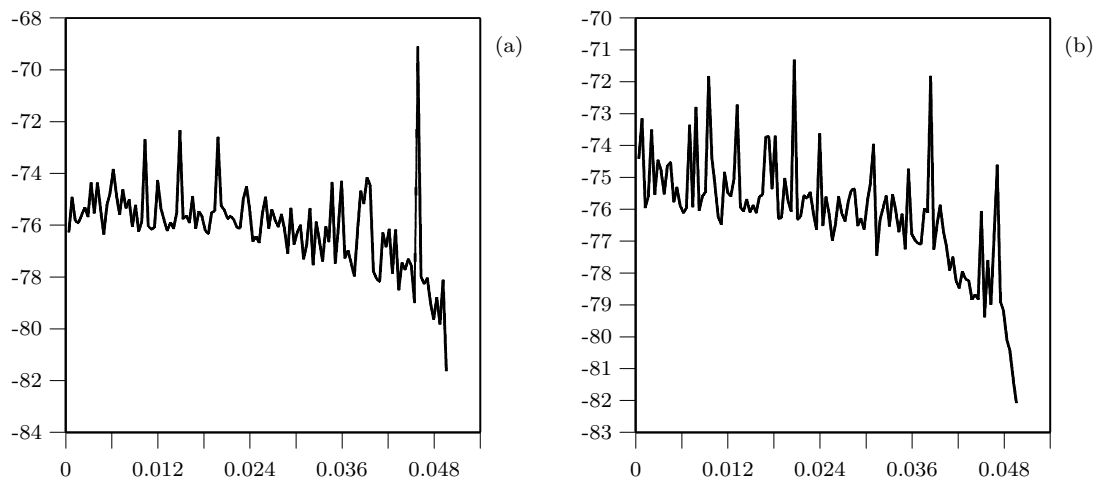


Figure 3.10.1. Deux estimations de la vraisemblance avec 1M d'itérations pour 120 valeurs conductrices pour l'exemple C.

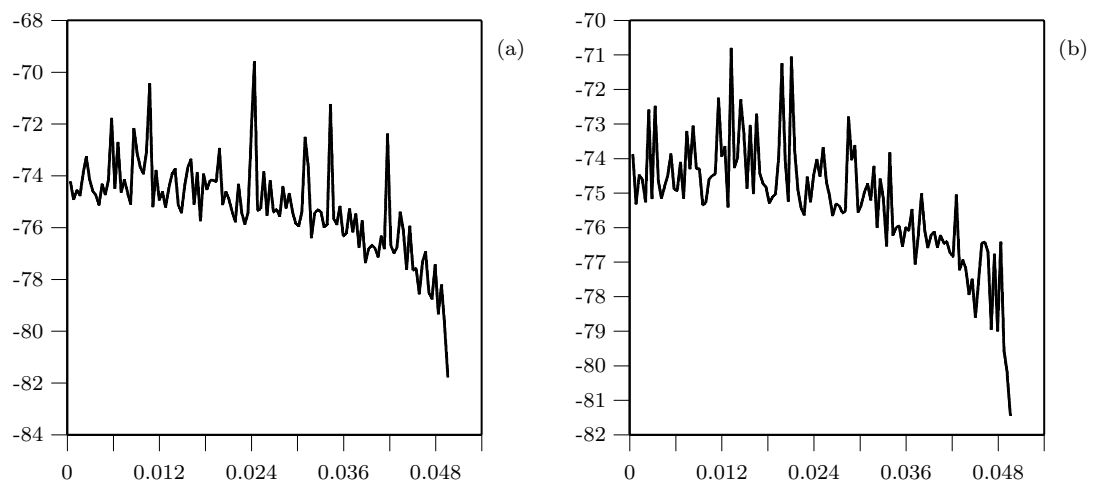


Figure 3.10.2. Deux estimations de la vraisemblance avec 2M d'itérations pour 120 valeurs conductrices pour l'exemple C.

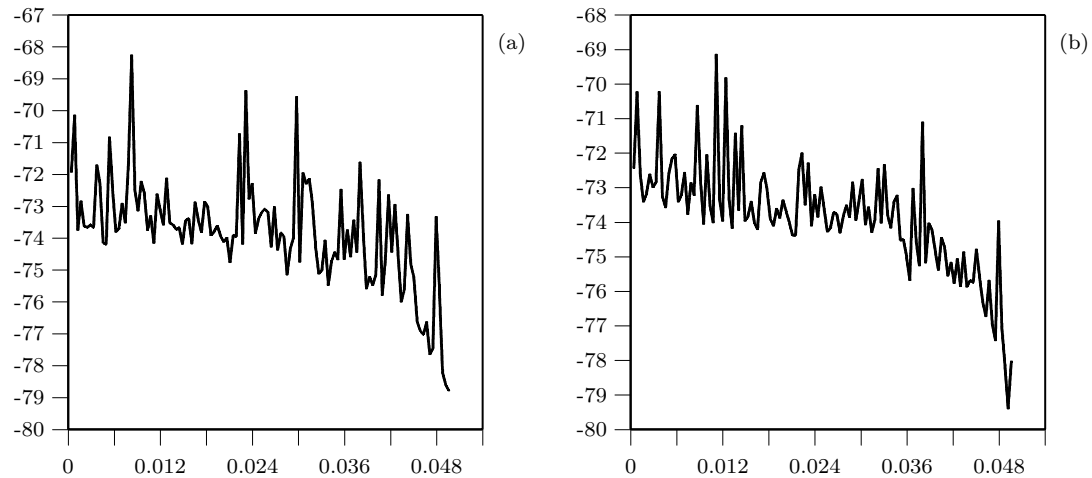


Figure 3.10.3. Deux estimations de la vraisemblance avec 5M d'itérations pour 120 valeurs conductrices pour l'exemple C.

À titre informatif, une estimation avec 5M d'itérations pour cet exemple avec 120 valeurs conductrices nécessite plus de deux jours de calcul sur un ordinateur muni d'un Pentium III 1GHz. Notons que pour une estimation comme celle que nous présentons ici, quand on parle de 1M d'itérations par exemple, cela signifie que 1M d'itérations sont faites pour chacune des 120 valeurs conductrices, ce qui fait que 120M de graphes ont été évalués pour obtenir le profil montré. Ces graphes étant légèrement différents de ceux que nous avons présenté jusqu'à présent, précisons que, comme d'habitude, l'axe des ordonnées représente le logarithme de $L(r_T)$, et que l'axe des abscisses représente r_T ; la seule différence est que l'évaluation de la vraisemblance à $r_T = x$ est effectuée avec la valeur conductrice $r_{Tp} = x$.

Rappelons que la vraie valeur de r_T est ici d'environ 0.01, soit très proche du début de la séquence. Quel que soit le nombre d'itérations, on remarque que les profils sont assez semblables: la vraisemblance est plus élevée au début de la séquence. Certaines courbes montrent un maximum local entre 0.006 et 0.024, mais il n'y a pas d'évidence que la vraisemblance augmente de façon marquée autour de la vraie valeur de r_T . L'allure générale des profils est très similaire aux profils obtenus avec une seule valeur conductrice, tel qu'illustré à la figure 3.9.4.

Voyons maintenant le même type d'évaluation de la vraisemblance pour l'exemple B. Les figures 3.10.4, 3.10.5 et 3.10.6 montrent respectivement des esti-

mations de la vraisemblance avec 1M, 5M et 10M d'itérations, faites sur le même modèle que l'exemple précédent. Pour cet exemple, 100 valeurs conductrices ont été utilisées réparties sur les trois intervalles de la séquence. Il faut un peu plus de sept jours de calcul pour faire une estimation de 10M d'itérations. Ici encore, les profils de vraisemblance sont assez similaires quel que soit le nombre d'itérations utilisé. Le maximum des profils est toujours vers le milieu de la séquence.

Seul un profil avec 5M d'itérations et un autre avec 10M d'itérations montrent un élévation marquée de la vraisemblance proche de la vraie valeur de r_T .

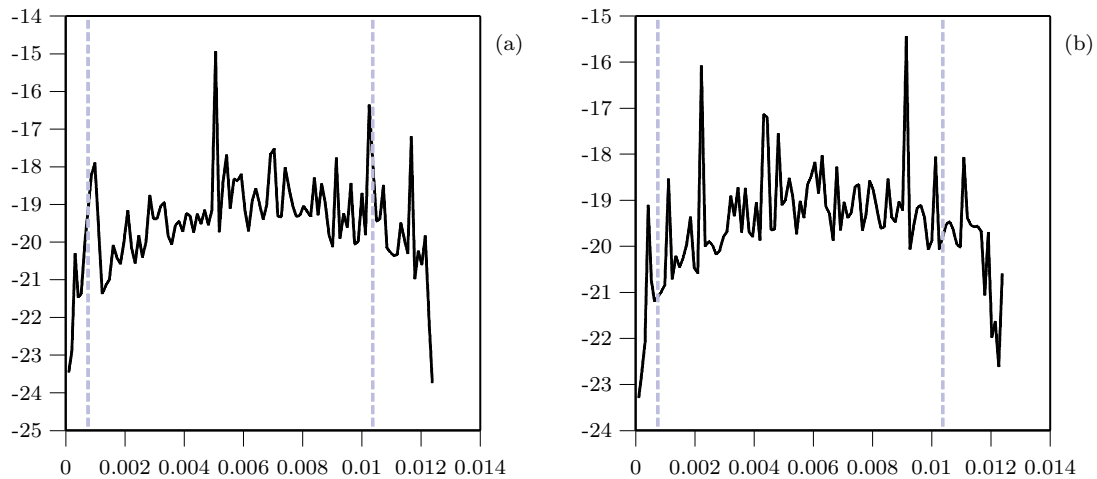


Figure 3.10.4. Deux estimations de la vraisemblance avec 1M d'itérations pour 100 valeurs conductrices pour l'exemple B.

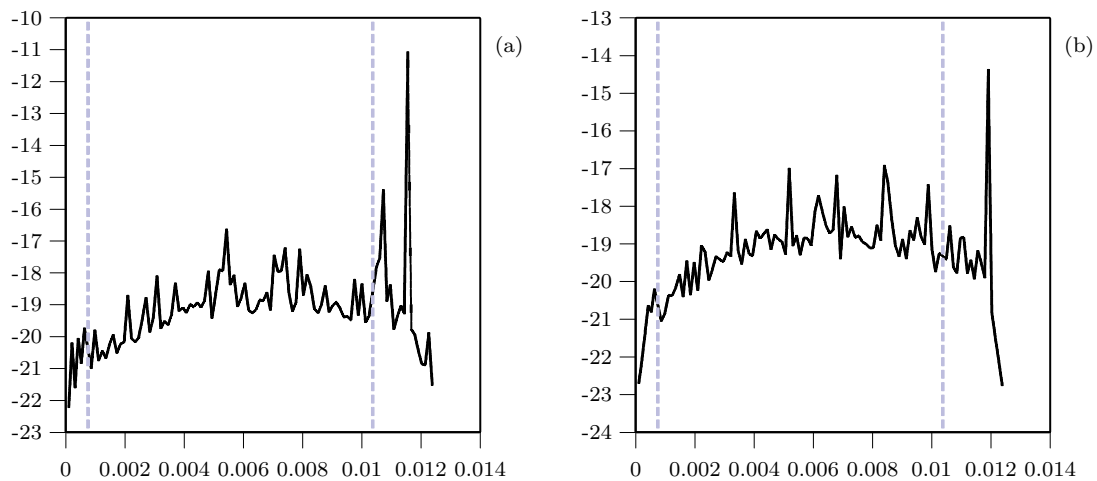


Figure 3.10.5. Deux estimations de la vraisemblance avec 5M d'itérations pour 100 valeurs conductrices pour l'exemple B.

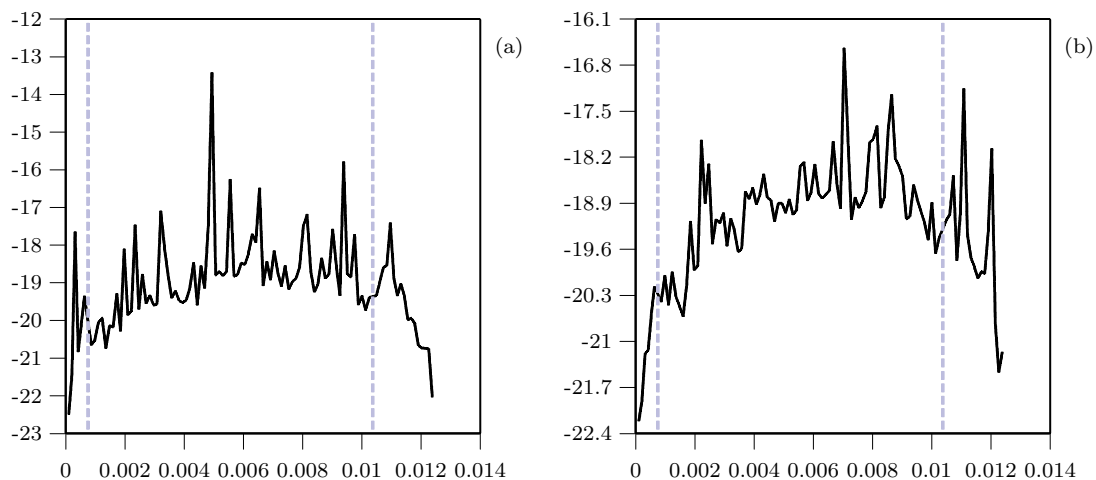


Figure 3.10.6. Deux estimations de la vraisemblance avec 10M d'itérations pour 100 valeurs conductrices pour l'exemple B.

Ici encore, les profils sont assez similaires aux profils que nous avons obtenus précédemment (fig. 3.9.3). Pour conclure cette section, on peut voir à partir des exemples que nous avons présentés que la fonction estimant la vraisemblance pour les autres points que la valeur conductrice, au centre d'un intervalle entre marqueurs, semble bien fonctionner. Les profils que l'on observe ici, avec une centaine de valeurs conductrices, sont très similaires aux profils générés par la méthode en utilisant une seule valeur conductrice. L'utilisation de plusieurs valeurs conductrices, comme le font Fearnhead et Donnelly (2001), ne semble pas être une nécessité, bien qu'il est évident que son utilisation ne peut nuire aux estimations. Notons cependant qu'un tel procédé n'est pas facile à installer, et les calculs sont alourdis.

3.11. Méthode approximative pour les recombinaisons

Dans la reconstruction du graphe de recombinaison ancestral, nous devons calculer, à chaque étape du processus, les probabilités de tous les événements qui ont pu se produire un pas en arrière dans le temps. Pour gagner du temps, les événements de recombinaison peuvent être simulés globalement pour toutes les séquences, au lieu d'être simulés un à un, c'est-à-dire pour chaque séquence et chaque intervalle possible. Pour simuler un événement de recombinaison, on peut

d'abord choisir une séquence au hasard et ensuite un intervalle de recombinaison au hasard pour la séquence choisie, comme dans Griffiths et Marjoram (1996), ou de façon équivalente, utiliser la méthode suivante: choisissons d'abord un intervalle de recombinaison au hasard, et ensuite une séquence au hasard qui va recombiner au point choisi. De cette façon, moins d'événements sont à prendre en compte. Par exemple, si nous prenons l'exemple D, au début de la reconstruction du graphe, nous avons 10 types différents de séquences et cinq intervalles entre loci, donc 50 événements de recombinaison à considérer; avec la modification proposée ici, seuls 5 événements de recombinaison seront à prendre en compte: un par intervalle entre marqueurs. Nous allons donc considérer le nombre de séquences pour lesquelles l'intervalle p est ancestral pour une recombinaison, nombre que nous noterons $n(p)$. Ceci est le cas pour une séquence de type i si l'intervalle p est entre le premier marqueur (γ_i) et le dernier marqueur (κ_i) dans l'ensemble des marqueurs ancestraux, c'est-à-dire $n(p) = \sum_i \delta_i^p$, où $\delta_i^p = 1$ si $\gamma_i \leq p \leq \kappa_i$, et $\delta_i^p = 0$ sinon. Nous pouvons alors réécrire l'équation de récurrence de la section 3.4 comme suit:

$$\begin{aligned}
Q(\mathbf{H}_\tau) = \frac{1}{D_{\mathbf{H}_\tau}} & \left[n \sum_{\substack{i, \\ n_i > 1}} (n_i - 1) Q(\mathbf{H}_\tau + C_i) \right. \\
& + 2n \sum_{\substack{i \neq j \\ \text{compatibles}}} (n_k + 1 - \delta_{ik} - \delta_{jk}) Q(\mathbf{H}_\tau + C_{ij}^k) \\
& + \sum_i \sum_{\substack{m \in A_i, s_m^i \neq s_m^j \\ \text{pour tout } k \neq i}} \theta(n_j + 1)/L Q(\mathbf{H}_\tau + M_i^j(m)) \\
& \left. + \frac{1}{n+1} \sum_{p=1}^{L-1} n(p) \rho_p(n_J + 1)(n_K + 1) Q(\mathbf{H}_\tau + R_I^{JK}(p)) \right],
\end{aligned}$$

où I est une séquence choisie au hasard, parmi toutes celles dont l'intervalle p est ancestral, et les séquences J et K sont déterminées par I et p . Le dernier terme de l'équation ci-dessus peut, de façon équivalente, s'écrire de la façon suivante:

$$\frac{1}{n+1} \sum_{p=1}^{L-1} n(p) \left[\rho_p (1 - \delta_p^l) (1 - \delta_p^r) + \rho_l \delta_p^l + \rho_r \delta_p^r \right] \\ \times (n_J + 1)(n_K + 1) Q(\mathbf{H}_\tau + R_I^{JK}(p)).$$

Par conséquent, on va choisir un intervalle de recombinaison au hasard, puis une séquence I au hasard, et finalement on calculera le facteur $(n_J + 1)(n_K + 1)$.

L'équation de récurrence devient alors:

$$Q(\mathbf{H}_\tau) = \sum_{\substack{i, \\ n_i > 1}} \frac{S_{\mathbf{H}_\tau}}{D_{\mathbf{H}_\tau}} \frac{n(n_i - 1)}{S_{\mathbf{H}_\tau}} Q(\mathbf{H}_\tau + C_i) \\ + \sum_{\substack{i \neq j \\ \text{compatibles}}} \frac{S_{\mathbf{H}_\tau}}{D_{\mathbf{H}_\tau}} \frac{2n(n_k + 1 - \delta_{ik} - \delta_{jk})}{S_{\mathbf{H}_\tau}} Q(\mathbf{H}_\tau + C_{ij}^k) \\ + \sum_i \sum_{\substack{m \in A_i, s_m^i \neq s_m^j \\ \text{pour tout } k \neq i}} \frac{S_{\mathbf{H}_\tau}}{D_{\mathbf{H}_\tau}} \frac{\theta(n_j + 1)}{L S_{\mathbf{H}_\tau}} Q(\mathbf{H}_\tau + M_i^j(m)) \\ + \sum_{p=1}^{L-1} n(p) \frac{(n_J + 1)(n_K + 1) S_{\mathbf{H}_\tau}}{D_{\mathbf{H}_\tau}} \frac{\rho_p}{(n+1) S_{\mathbf{H}_\tau}} Q(\mathbf{H}_\tau + R_I^{JK}(p)),$$

qui est de la forme

$$Q(\mathbf{H}_\tau) = \sum_{\mathbf{H}_{\tau+1}} f(\mathbf{H}_\tau, \mathbf{H}_{\tau+1}) P(\mathbf{H}_{\tau+1} | \mathbf{H}_\tau) Q(\mathbf{H}_{\tau+1}),$$

où $D_{\mathbf{H}_\tau} = [n(n-1) + n\alpha\theta + n\beta\rho]$ comme précédemment et

$$S_{\mathbf{H}_\tau} = n \sum_{\substack{i, \\ n_i > 1}} (n_i - 1) + 2n \sum_{\substack{i \neq j \\ \text{compatibles}}} (n_k + 1 - \delta_{ik} - \delta_{jk}) \\ + \sum_i \sum_{\substack{m \in A_i, s_m^i \neq s_m^j \\ \text{pour tout } k \neq i}} \theta(n_j + 1) / L + \sum_{p=1}^{L-1} n(p) \frac{\rho_p}{(n+1)}.$$

Au temps d'arrivée du $(\tau + 1)^{\text{ième}}$ événement, une transition est faite de \mathbf{H}_τ à:

$$(\mathbf{H}_\tau + C_i) \quad \text{avec probabilité} \quad n(n_i - 1) / S_{\mathbf{H}_\tau},$$

$$\begin{aligned}
(\mathbf{H}_\tau + C_{ij}^k) & \text{ avec probabilité } 2n(n_k + 1 - \delta_{ik} - \delta_{jk})/S_{\mathbf{H}_\tau}, \\
(\mathbf{H}_\tau + M_i^j(m)) & \text{ avec probabilité } \theta(n_j + 1)/(L S_{\mathbf{H}_\tau}), \\
(\mathbf{H}_\tau + R_I^{JK}(p)) & \text{ avec probabilité } [n(p) \cdot \rho_p]/[(n + 1)S_{\mathbf{H}_\tau}].
\end{aligned}$$

La fonction f est de la même forme que précédemment:

$$f(\mathbf{H}_\tau, \mathbf{H}_{\tau+1}) = \begin{cases} S_{\mathbf{H}_\tau}/D_{\mathbf{H}_\tau} & \text{si } Co \text{ ou } Mu \\ (n_J + 1)(n_K + 1)S_{\mathbf{H}_\tau}/D_{\mathbf{H}_\tau} & \text{si } Re, \end{cases}$$

alors que les fonctions h et ϕ diffèrent seulement en remplaçant les séquences i, j, k , en supposant un événement de recombinaison, par I, J, K , c'est-à-dire,

$$h_{\Theta_0}(\mathbf{H}_\tau, \mathbf{H}_{\tau+1}) = \frac{S_{(\mathbf{H}_\tau, \Theta_0)}}{D_{(\mathbf{H}_\tau, \Theta)}} \phi(\mathbf{H}_\tau, \mathbf{H}_{\tau+1}),$$

où

$$\phi(\mathbf{H}_\tau, \mathbf{H}_{\tau+1}) = \begin{cases} \theta/\theta_0 & \text{si } Mu \\ (n_J + 1)(n_K + 1) \frac{r_p(1-\delta_p^l)(1-\delta_p^r) + r_l\delta_p^l + r_r\delta_p^r}{r_p(1-\delta_p^l)(1-\delta_p^r) + r_{0l}\delta_p^l + r_{0r}\delta_p^r} & \text{si } Re \\ 1 & \text{si } Co. \end{cases}$$

Notons encore que si l'événement de recombinaison se produit dans un intervalle où le TIM n'est pas, alors $\delta_p^l = \delta_p^r = 0$, et donc $\phi(\mathbf{H}_\tau, \mathbf{H}_{\tau+1}) = (n_J + 1)(n_K + 1)$. Par ailleurs, $\phi(\mathbf{H}_\tau, \mathbf{H}_{\tau+1})$ est différent de 1 dès qu'un événement de recombinaison se produit dans un intervalle portant le TIM, i.e., lorsque δ_p^l ou δ_p^r sont différents de 0.

Nous allons maintenant comparer la méthode approximative ci-dessus et la méthode originale en ce qui concerne le temps de calcul et la variabilité des estimations, à l'aide des trois exemples A, B et C. Les résultats pour le premier exemple (A) sont présentés dans la figure 3.11.1. Les figures (a), (b) et (c) présentent les résultats avec la méthode originale, tandis que les figures (d), (e) et (f) présentent les résultats avec la méthode approximative. Les figures (a) et (d) sont basées sur 1M d'itérations, les figures (b) et (e) sont basées sur 5M d'itérations tandis que les figures (c) et (f) sont basées sur 10M d'itérations. Le temps nécessaire pour réaliser 1M d'itérations avec la méthode originale est d'environ 1h22m tandis

que ce temps est d'environ 43m avec la méthode approximative; la modification proposée dans cette section nous apporte, pour cet exemple, un gain de temps appréciable: les calculs sont presque deux fois plus rapides. L'examen de la figure 3.11.1 nous montre que les deux méthodes produisent bien des résultats similaires. Cependant, la méthode approximative donne un mauvais estimé global pour 1M et 10M d'itérations.

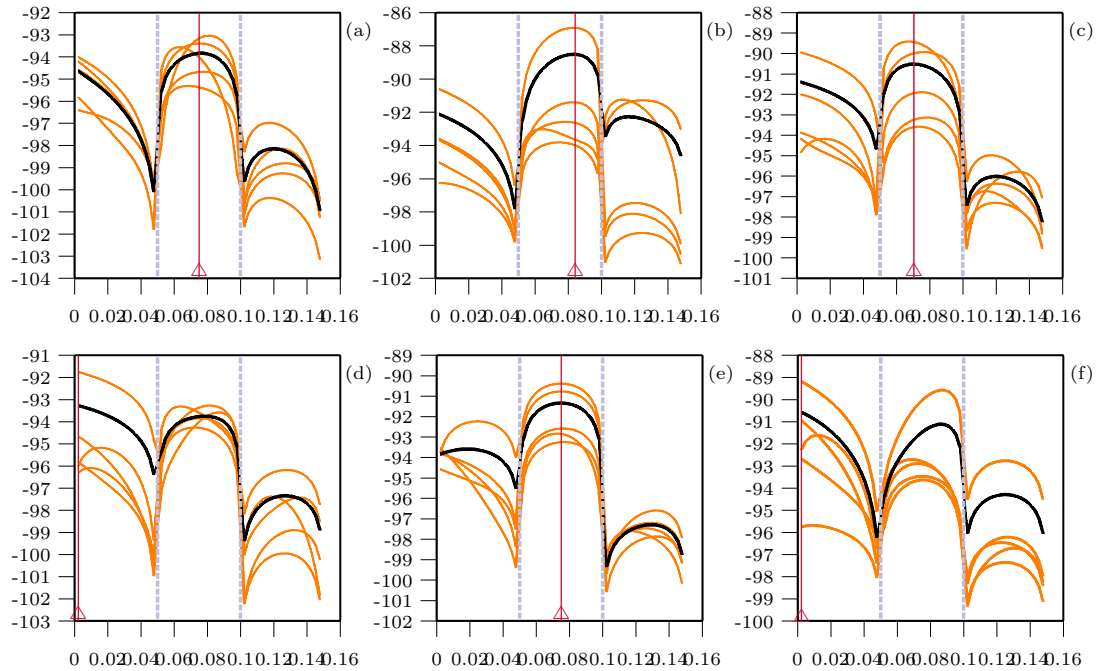


Figure 3.11.1. Courbes de vraisemblance pour l'exemple A: (a), (b) et (c) méthode originale; (d), (e) et (f) méthode approximative; (a) et (d) 1M d'itérations, (b) et (e) 5M d'itérations, (c) et (f) 10M d'itérations.

La figure 3.11.2 présente une comparaison similaire à celle que l'on vient de voir, mais pour l'exemple B. L'examen de la figure nous révèle que la variabilité est plus grande avec la méthode approximative. En fait, le portrait est légèrement différent de ce que nous montrait l'exemple précédent. On voit clairement, surtout pour les résultats obtenus à partir de 5M et de 20M d'itérations, qu'une courbe de vraisemblance parmi les cinq présente un profil assez différent de la tendance générale; si l'on excluait cette courbe, la variation ne serait pas plus grande qu'avec la méthode originale, et les estimations seraient tout à fait semblables. Évidemment, nous ne pouvons pas ignorer ces courbes aberrantes.

Le gain de temps est pour cet exemple d'un facteur deux: il faut 8m pour faire 1M d'itérations avec la méthode originale, tandis qu'il n'en faut que 4m avec la méthode approximative.

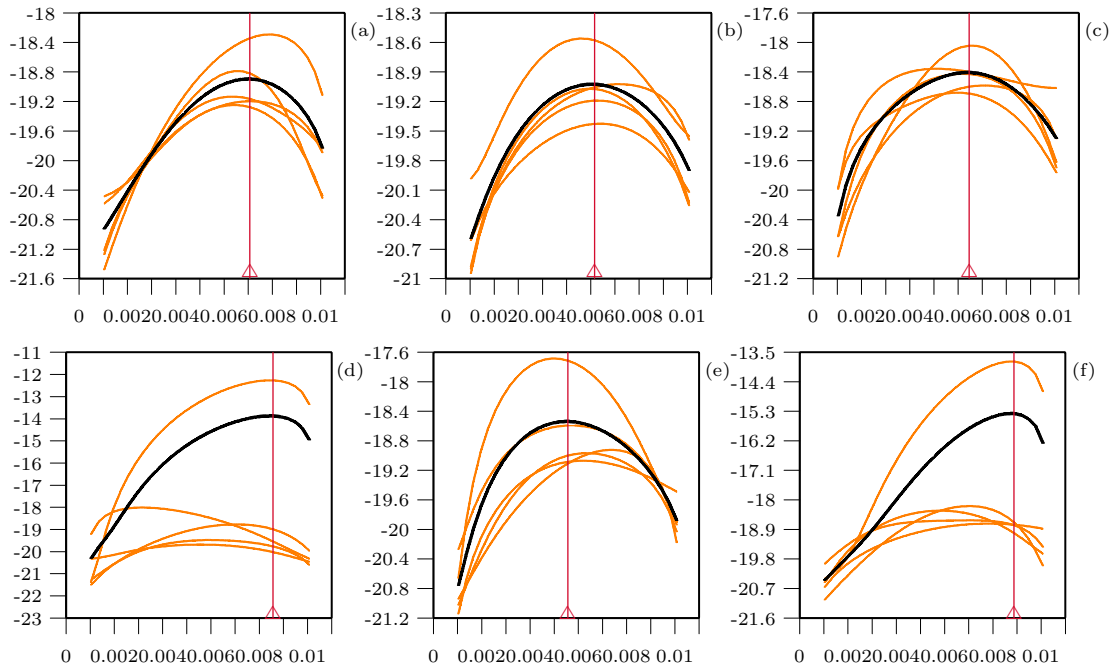


Figure 3.11.2. Courbes de vraisemblance pour l'exemple B: (a), (b) et (c) méthode originale; (d), (e) et (f) méthode approximative; (a) et (d) 1M d'itérations, (b) et (e) 5M d'itérations, (c) et (f) 20M d'itérations.

L'exemple C est étudié dans la figure 3.11.3. Bien qu'encore ici la variabilité avec la méthode approximative soit plus grande, elle se compare bien avec celle obtenue avec la méthode exacte pour 1M et 20M d'itérations. Par contre, dans le cas de 5M d'itérations, deux courbes de vraisemblance très différentes des autres peuvent être observées. Le gain de temps de calcul est moins grand pour cet exemple: il est d'un facteur 1.5; il faut 10m pour réaliser 1M d'itérations avec la méthode exacte, tandis que seulement 6m sont nécessaires pour la méthode approximative.

Une des raisons qui pourraient expliquer ces courbes aberrantes est que la fonction $f(\mathbf{H}_\tau, \mathbf{H}_{\tau+1}) = S_{\mathbf{H}_\tau} / D_{\mathbf{H}_\tau}$ est multipliée par $(n_J + 1)(n_K + 1)$ dans la méthode approximative quand une recombinaison se produit, et qu'il arrive peut-être parfois que n_J ou n_K présentent des valeurs qui donnent un poids très élevé

à un graphe en particulier. Le facteur équivalent à $(n_J + 1)(n_K + 1)$ est pour la méthode originale $(n_j + 1)(n_k + 1)$, mais il n'entre pas directement dans le calcul de $f(\mathbf{H}_\tau, \mathbf{H}_{\tau+1})$; il intervient dans le calcul de la probabilité de recombinaison.

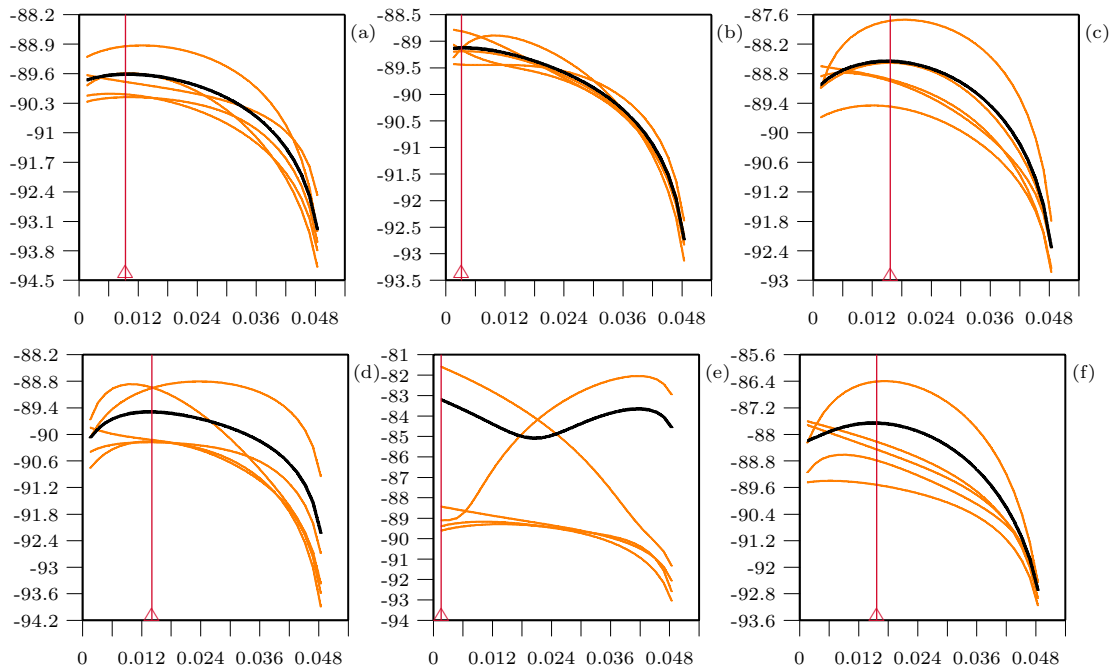


Figure 3.11.3. Courbes de vraisemblance pour l'exemple C: (a), (b) et (c) méthode originale; (d), (e) et (f) méthode approximative; (a) et (d) 1M d'itérations, (b) et (e) 10M d'itérations, (c) et (f) 20M d'itérations.

Enfin, il est intéressant de noter que dans l'implantation de Griffiths et Marjoram (1996), un schéma similaire est employé, mais les auteurs considèrent la méthode originale comme un pas intermédiaire vers le développement d'une méthode approximative similaire à celle que nous présentons ici, et ne l'utilisent pas. La raison probable est que les auteurs considèrent la reconstruction de séquences de nucléotides, donc l'équivalent d'un grand nombre de marqueurs dans nos termes; il est alors non seulement coûteux, mais parfois presque impossible d'utiliser la méthode exacte. Il semble pourtant d'après nos résultats que la méthode pleine, bien que demandant plus de temps de calcul, montre moins de variation.

3.12. Taille de population variable

Nous avons supposé jusqu'à maintenant que la taille de la population était constante. Pour des applications réalistes par contre, il est judicieux de permettre une taille de population variable. Dans le processus de coalescence sans recombinaison (Griffiths et Tavaré, 1997, Donnelly et Tavaré, 1995, Nordborg, 2001), le changement dans la taille de la population cause simplement un changement d'échelle dans l'arbre de coalescence. Par exemple, si la population a une croissance exponentielle en avançant dans le temps, alors plus on remonte dans le temps, plus un événement de coalescence est probable (par rapport aux autres événements), car la population est plus petite et cela prend moins de temps pour qu'une coalescence se produise. Avec le phénomène de recombinaison, une taille de population variable ne change pas seulement l'échelle du graphe de recombinaison ancestral, mais aussi sa topologie. Dans le cas d'une croissance de population exponentielle, par exemple, plus nous remontons dans le temps, plus un événement de coalescence est susceptible de se produire par rapport à un événement de recombinaison.

Soit W_τ le temps où se produit le $\tau^{\text{ième}}$ événement en remontant dans le temps ($\tau = 0, \dots, \tau^*$), et $S_\tau = W_\tau - W_{\tau-1}$. Si la taille de la population est constante et donnée par $2N$ et que le nombre de séquences ancestrales avant l'arrivée du $(\tau + 1)^{\text{ième}}$ événement est n , avec une proportion moyenne de loci ancestraux par séquence α et une proportion moyenne de la longueur d'une séquence couverte par des intervalles ancestraux β , alors les probabilités de coalescence, recombinaison et mutation affectant le matériel ancestral en une génération vers le passé sont, approximativement et respectivement, $\binom{n}{2}/(2N)$, $n\beta r$ et $n\alpha u$, et la probabilité qu'au moins un de ces événements se produise est approximativement

$$\begin{aligned} & 1 - \left(1 - \frac{n(n-1)}{4N}\right) (1 - n\beta r) (1 - n\alpha u) \\ & \approx \frac{n}{4N} (n-1 + \alpha\theta + \beta\rho). \end{aligned}$$

Ces approximations sont obtenues en ne retenant que les termes d'ordre $1/N$.

En prenant $2N$ générations comme unités de temps, et en faisant tendre N vers l'infini, le temps jusqu'au prochain événement (le $(\tau + 1)^{\text{ième}}$) suit une loi exponentielle de paramètre $n(n - 1 + \alpha\theta + \beta\rho)/2$. Donc, on a (Griffiths et Marjoram, 1996):

$$E(S_{\tau+1}|W_\tau) = 2/[n(n - 1 + \alpha\theta + \beta\rho)]. \quad (3.6)$$

Supposons maintenant que la taille de la population varie dans le temps d'une façon déterministe. Nous supposons que nous connaissons $2N(t)$, la taille de la population au temps t dans le passé mesuré en unités de $2N(0)$ générations. Dénotons par $\nu(t)$ le ratio de la taille de la population au temps t sur celui au temps 0, soit $\nu(t) = N(t)/N(0)$, où $N(0) = N$. Définissons $\lambda(t) = 1/\nu(t)$. Les probabilités de coalescence, recombinaison et mutation précédentes deviennent alors respectivement $[n(n - 1)\lambda(t)]/[4N]$, $n\beta r$ et $n\alpha$. Donc, les probabilités de ces événements sachant qu'au moins un d'entre eux se produit sont, lorsque N tend vers l'infini:

$$\begin{aligned} P(\text{Co}|\cdot) &= \frac{n(n - 1)\lambda(t)}{n(n - 1)\lambda(t) + n\beta\rho + n\alpha\theta}, \\ P(\text{Re}|\cdot) &= \frac{n\beta\rho}{n(n - 1)\lambda(t) + n\beta\rho + n\alpha\theta}, \\ P(\text{Mu}|\cdot) &= \frac{n\alpha\theta}{n(n - 1)\lambda(t) + n\beta\rho + n\alpha\theta}. \end{aligned}$$

Alors, l'équation de récurrence pour $Q(\mathbf{H}_\tau)$ qui affecte le matériel ancestral devient:

$$\begin{aligned} Q(\mathbf{H}_\tau) &= \\ &\int_{W_\tau}^{\infty} \left[\frac{n\lambda(W_{\tau+1})}{D(W_{\tau+1})} \sum_{\substack{i, \\ n_i > 1}} (n_i - 1) Q(\mathbf{H}_\tau + C_i) \right. \\ &\quad \left. + \frac{2n\lambda(W_{\tau+1})}{D(W_{\tau+1})} \sum_{\substack{i \neq j \\ \text{compatibles}}} \frac{n_k + 1 - \delta_{ik} - \delta_{jk}}{n - 1} Q(\mathbf{H}_\tau + C_{ij}^k) \right] \end{aligned}$$

$$\begin{aligned}
& + \frac{1}{D(W_{\tau+1})} \sum_i \sum_{\substack{m \in A_i, s_m^i \neq s_m^j \\ \text{pour tout } k \neq i}} \theta \frac{n_j + 1}{L n} Q(\mathbf{H}_\tau + M_i^j(m)) \\
& + \frac{1}{D(W_{\tau+1})} \sum_{p=1}^{L-1} n(p) \rho_p \frac{(n_j + 1)(n_k + 1)}{n(n+1)} Q(\mathbf{H}_\tau + R_i^{jk}(p)) \Big] g(W_{\tau+1}|W_\tau) dW_{\tau+1},
\end{aligned}$$

où $D(W_{\tau+1}) = [n(n-1)\lambda(W_{\tau+1}) + n\alpha\theta + n\beta\rho]$, et $g(W_{\tau+1}|W_\tau)$ est la distribution du temps où se produit le $(\tau+1)$ ^{ième} événement, sachant le temps où s'est produit τ ^{ième} événement. La probabilité que $W_{\tau+1}$ soit plus grand qu'une valeur $w_{\tau+1}$ sachant $W_\tau = w_\tau$ est la probabilité qu'aucun événement de coalescence, recombinaison ou mutation ne se produise dans l'intervalle de temps $S_{\tau+1}$, et est donnée par:

$$P(W_{\tau+1} > w_{\tau+1} | W_\tau = w_\tau) = \exp\left(-\int_{w_\tau}^{w_{\tau+1}} \Phi(u) du\right),$$

où

$$\Phi(u) = \frac{n(n-1)}{2} \lambda(u) + \frac{n\alpha\theta}{2} + \frac{n\beta\rho}{2}$$

est le taux de changement dans le matériel ancestral. Maintenant, pour tenir compte du temps et de la taille de population variable dans l'algorithme de Monte-Carlo, on doit simuler le temps du prochain événement selon une distribution appropriée (Donnelly et Tavaré, 1995; Griffiths et Tavaré, 1996). Supposons que $\lambda(u) = \exp(\kappa u)$, de telle sorte que

$$N(t) = N(0) \exp(-\kappa u),$$

c'est-à-dire que la taille de la population décroît exponentiellement vers le passé. Donnelly et Tavaré (1995) ont suggéré de simuler les temps d'arrivée des événements de la façon suivante: on considère $\{U_0, U_1, \dots\}$ une séquence de variables aléatoires mutuellement indépendantes distribuées uniformément et, en utilisant le fait que 1 moins la fonction de répartition de $W_{\tau+1}$ évaluée à $W_{\tau+1}$ est distribuée uniformément, on résout l'équation:

$$\exp\left(-\int_{w_\tau}^{w_{\tau+1}} \Phi(u) du\right) = u_\tau,$$

qui donne:

$$-\int_{w_\tau}^{w_{\tau+1}} \left\{ \frac{n(n-1)}{2} \exp(\kappa u) + \frac{n\alpha\theta}{2} + \frac{n\beta\rho}{2} \right\} du = \log(u_\tau),$$

c'est-à-dire:

$$\begin{aligned} & -\frac{n(n-1)}{2\kappa} \exp(\kappa w_{\tau+1}) - w_{\tau+1} \left(\frac{n\alpha\theta}{2} + \frac{n\beta\rho}{2} \right) \\ & + \frac{n(n-1)}{2\kappa} \exp(\kappa w_\tau) + w_\tau \left(\frac{n\alpha\theta}{2} + \frac{n\beta\rho}{2} \right) - \log(u_\tau) = 0. \end{aligned}$$

Il n'y a pas de solution directe pour exprimer la variable inconnue $w_{\tau+1}$ en fonction des autres paramètres. Cependant, dans le cas particulier où $\theta = \rho = 0$, l'équation ci-dessus devient:

$$w_{\tau+1} = \frac{1}{\kappa} \log \left[\frac{-2\kappa}{n(n-1)} \log(u_\tau) + \exp(\kappa w_\tau) \right]. \quad (3.7)$$

Ce résultat peut être trouvé dans Griffiths et Tavaré (1998) et Griffiths (2001). Maintenant, si nous revenons au cas général, nous devons trouver la solution d'une équation de la forme:

$$a \exp(bs) + sc + d = 0,$$

qui peut être trouvée en utilisant le logiciel de calcul symbolique Maple:

$$s = -\frac{1}{bc} \left[W \left(\frac{1}{c} ab \exp \left(-\frac{db}{c} \right) \right) c + db \right], \quad (3.8)$$

où W est la fonction W de *Lambert* définie par:

$$W(x) = \sum_{n=1}^{\infty} \frac{(-1)^{n-1} n^{n-2}}{(n-1)!} x^n.$$

Cette fonction est l'inverse de la fonction we^w (Corless *et al.*, 1996). Un algorithme basé sur les itérations de Halley converge rapidement pour tous les x valides (Briggs, 1998). L'algorithme produit la valeur $w = W(x)$. En utilisant

cette stratégie, on peut maintenant modéliser la décroissance exponentielle de la population en reculant dans le temps. L'algorithme est le suivant:

```

si  $i < 1/e$  alors fin;
si  $x = 0$  alors  $w = 0$ ;
/* approximation initiale */
 $w = \ln(x)$ ;
si  $x < 1$  alors faire;
     $p = \sqrt{2 * (e * x + 1)}$ ;
     $w = -1 + p - p^2/3 + 11/72 * p^3$ ;
fin;
si  $x > 3$  alors  $w = w - \ln(w)$ ;
faire pour  $I = 0$  à 20;
     $e = \exp(w)$ ;
     $t = w * e - x$ ;
     $t = t / (e * (w + 1) - 0.5 * (w + 2) * t / (w + 1))$ ;
     $w = w - t$ ;
    si  $|t| < precision * (1 + |w|)$  alors fin;
fin;
```

où e est la constante d'Euler. Montrons maintenant un exemple pour la distribution du temps au prochain événement. Nous avons simulé 80000 valeurs aléatoires d'une distribution $U(0, 1)$, et les observations ont été regroupées pour obtenir une approximation de la densité des s_c (coalescence seulement) en utilisant l'équation (3.7) et s_{crm} (coalescence, recombinaison et mutation) en utilisant l'équation (3.8), et les moyennes correspondantes μ_c et μ_{crm} . Trois cas ont été considérés, avec des ensembles de paramètres différents:

- (a) $n = 140, \kappa = 1700, \theta = 10, \rho = 10$ et $t = 0.00001$.
On obtient alors $\mu_c = 2.01 \times 10^{-4}$ et $\mu_{crm} = 1.78 \times 10^{-4}$.
- (b) $n = 140, \kappa = 1700, \theta = 100, \rho = 100$ et $t = 0.00001$.
On obtient alors $\mu_c = 2.01 \times 10^{-4}$ et $\mu_{crm} = 8.55 \times 10^{-5}$.
- (c) $n = 140, \kappa = 1700, \theta = 358, \rho = 358$ et $t = 0.00001$.
On obtient alors $\mu_c = 2.01 \times 10^{-4}$ et $\mu_{crm} = 3.81 \times 10^{-5}$.

Les densités pour les trois cas sont illustrées dans la figure 3.12.1.

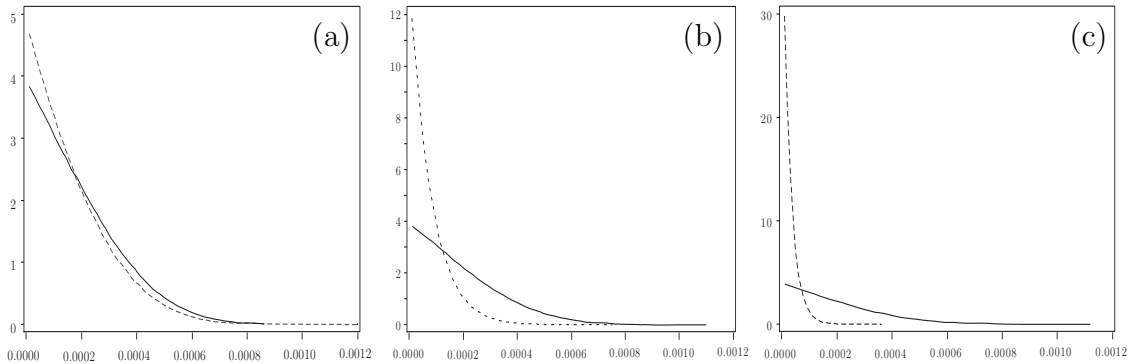


Figure 3.12.1. Densités pour les trois cas (a), (b) et (c). La ligne pleine est pour s_c et la ligne hachurée pour s_{crm} .

Dans le premier cas (a), où θ et ρ sont petits, les deux densités sont presque identiques. Dans le second cas (b), on peut voir une différence importante dans les distributions: le prochain événement arrivera plus tôt quand on tient compte des recombinaisons. Enfin, le dernier cas (c) illustre la situation où les taux de recombinaison et de mutation sont grands (correspondant en fait au cas de l'exemple D, cf. Virtaneva *et al.*, 1996).

Comme souligné par R.C. Griffiths (communication personnelle), il existe un autre moyen de simuler le temps du prochain événement. On simule deux variables indépendantes X et Y tel que:

$$P(X > x) = \exp\left(-\int_{w_\tau}^x \binom{n}{2} \exp(\kappa u) du\right)$$

$$P(Y > y) = \exp\left(-\frac{n}{2}(y - w_\tau)(\alpha\theta + \beta\rho)\right).$$

On prend alors $W_{\tau+1} = \min(X, Y)$.

Plus θ et ρ sont grands, plus petit sera le temps d'attente du prochain événement; il devient alors plus important de tenir compte des tailles de populations variables. Une grande valeur du taux de recombinaison est spécialement importante si l'on veut étudier des séquences de longueur modérée. Nous utiliserons ces résultats quand nous étudierons les exemples D et E dans le chapitre 5.

Chapitre 4

Implantation informatique

La méthode proposée dans ce travail utilise des simulations de Monte-Carlo. À ce titre, l'aspect informatique joue un rôle important dans le développement de la méthode. Nous allons décrire dans ce chapitre les aspects informatiques de la méthode: le langage de programmation utilisé et l'algorithme développé.

4.1. Choix du langage de programmation

La méthodologie présentée a été programmée en C++ standard, en utilisant certaines fonctionnalités de la STL (Standard Template Library). Ce langage de programmation possède l'avantage d'être facilement portable s'il est écrit en C++ standard, de pouvoir utiliser les nombreuses extensions de la STL qui sont disponibles publiquement, et de fonctionner sur la plupart des systèmes informatiques. De plus, le C++ est un langage très efficace en ce qui concerne la rapidité d'exécution. Pour faciliter sa portabilité, seule une interface textuelle est disponible. Ce programme compile correctement, sans aucune modification, dans

les environnements Windows, Solaris, et Linux, et avec quelques modifications mineures également sous Irix, tout cela avec la version GNU de C++. Le programme comporte environ 4000 lignes de codes; il est présenté dans l'annexe C.

4.2. Algorithme

Étant donné que le graphe de recombinaison ancestral est un processus markovien, il est assez simple de programmer sa reconstruction. Une fois les données et les paramètres lus, il suffit d'insérer le TIM dans le premier intervalle entre marqueurs, de construire K graphes afin d'évaluer la vraisemblance sur cet intervalle, et de continuer ainsi pour chaque intervalle. Pour construire un graphe, il suffit d'étudier les événements possibles, de calculer leur probabilité, de choisir un événement au hasard, de calculer et mémoriser le poids de l'événement pour la vraisemblance, de mettre à jour les données et les paramètres, et de recommencer jusqu'à ce que l'ancêtre commun de toutes les séquences soit trouvé. Enfin, il suffit de chercher le maximum de vraisemblance, et d'imprimer les résultats. Nous allons maintenant décrire plus formellement l'algorithme. La notation "f()" est utilisée pour dénoter une fonction f qui dépend de certains paramètres.

```
// Algorithme
début{
  LireDonnées()           lecture des données
  LireParamètres()       lecture des paramètres
  pour ( $p=1$  à  $L-1$ ) {     pour chacun des intervalles
    créer  $Dist[]$          modification du vecteur des distances
    créer  $Y[]$             modification de la matrice
                           des haplotypes
    pour ( $k=1$  à  $K$ ) {    pour chacune des  $K$  simulations
      tant que ( $n>1$ ) {  tant que le MRCA n'est pas trouvé
        PossCoaId()      étudier les coalescences de types
                           identiques possibles
        PossCoaDi()      étudier les coalescences de types
```

	<i>différents possibles</i>
PossMut()	<i>étudier les mutations possibles</i>
PossRecom()	<i>étudier les recombinaisons possibles</i>
ChoisirÉvénement ()	<i>choisir un évènement</i>
MiseÀJour()	<i>mise à jour des données</i>
}	
}	
}	
EcrireResultat()	<i>écriture des résultats</i>
}	
Fin	

Le programme commence par lire le nom du fichier contenant les paramètres, entré sur la ligne de commande. Dans ce fichier de paramètres, le programme lit les informations suivantes:

- 1) le nom du fichier de données,
- 2) u , le taux de mutation,
- 3) r_1, r_2, \dots, r_{L-1} , les distances entre marqueurs (de position connue),
- 4) K , le nombre de simulations,
- 5) N , la taille effective de la population,
- 6) ξ , le poids que l'on accorde aux événements de recombinaison,
- 7) κ , le paramètre de la distribution exponentielle pour une taille de population variable,
- 8) le nombre de points pour lesquels on veut évaluer la vraisemblance,
- 9) les éventuels nombres de points spécifiés par intervalle,
- 10) le nom du fichier dans lequel les résultats seront écrits,
- 11) ν , le nombre maximum d'itérations dans un graphe,
- 12) le choix des recombinaisons: dans chaque séquence et chaque intervalle, ou en utilisant l'approximation de la section 3.11.

Les données lues sont les haplotypes (\mathbf{Y}'), le trait (\mathbf{T}), la multiplicité des séquences (\mathbf{n}), et les distances entre marqueurs (\mathbf{r}'), dont voici un exemple:

$$\mathbf{Y}' = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{T} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}, \quad \mathbf{n} = \begin{bmatrix} 2 \\ 3 \\ 1 \\ 2 \end{bmatrix}, \quad \mathbf{r}' = [0.02, 0.06].$$

Dans la matrice \mathbf{Y}' , de dimension $d \times (L - 1)$, chaque ligne est une séquence, et chaque colonne un marqueur. Ainsi, par exemple, la première séquence est $(0, 0, 0)$ et la seconde $(1, 0, 0)$. Notons que la représentation informatique que nous utilisons pour un site primitif ancestral est un 0, pour un site mutant un 1 et pour un site non ancestral un 9, mais dans le texte qui suit nous représentons un site non ancestral par le symbole “●” afin de faciliter la lecture. La $i^{\text{ième}}$ séquence de \mathbf{Y}' est un cas si $\mathbf{T}_i = 1$, et un témoin si $\mathbf{T}_i = 0$. Aussi, la $i^{\text{ième}}$ séquence de \mathbf{Y}' a pour multiplicité \mathbf{n}_i (le $i^{\text{ième}}$ élément de \mathbf{n}). Ainsi, la quatrième séquence de \mathbf{Y}' qui est $(0, 0, 1)$ est un cas et a pour multiplicité 2. Enfin, puisque nous avons trois marqueurs, nous avons deux distances entre marqueurs (voir \mathbf{r}'): la distance entre le premier et le second marqueur est 0.02, et la distance entre le second et le troisième est 0.06.

Une fois les données et les paramètres lus, pour chaque intervalle entre marqueurs p ($p = 1, \dots, L - 2$), on modifie les haplotypes et le vecteur de distance:

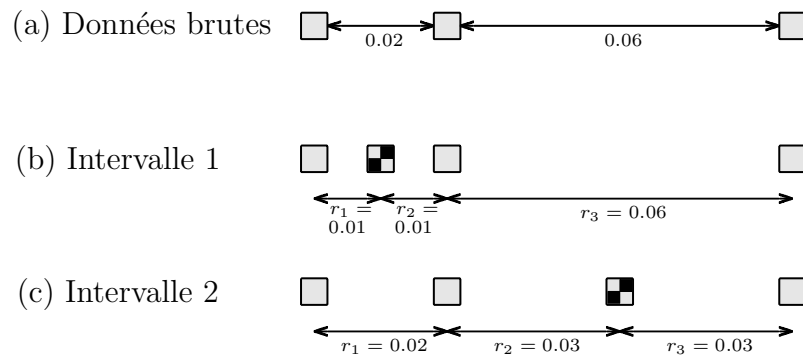


Figure 4.2.1. Schéma d’insertion du TIM (■) dans la séquence des marqueurs observés de position connue (□). Les chiffres indiquent des distances entre loci. (a) Données brutes: 3 marqueurs. Modification des séquences et des distances pour l’évaluation de la vraisemblance dans l’intervalle entre marqueurs (b) un (c) deux.

pour l'intervalle p , on insère dans chacune des d séquences de \mathbf{Y}' un nouveau marqueur, dont la valeur se trouve dans le vecteur \mathbf{T} , exactement entre les marqueurs p et $p + 1$. Il faut en conséquence ajuster le vecteur des distances en divisant la distance r_p existante en deux (car la valeur conductrice est au milieu de l'intervalle entre marqueur p), et en rajoutant la même distance après l'élément p du vecteur \mathbf{r}' , comme illustré dans la figure 4.2.1.

Voici un exemple, où l'on travaille sur l'intervalle 2 ($p = 2$) avec les données ci-dessus:

$$\mathbf{r}' = \begin{bmatrix} 0.02 \\ 0.06 \end{bmatrix}, \quad \mathbf{Y}' = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

$$\mathbf{r} = \begin{bmatrix} 0.02 \\ \mathbf{0.03} \\ \mathbf{0.03} \end{bmatrix}, \quad \mathbf{Y} = \begin{bmatrix} 0 & 0 & \mathbf{0} & 0 \\ 1 & 0 & \mathbf{0} & 0 \\ 1 & 0 & \mathbf{1} & 0 \\ 0 & 0 & \mathbf{1} & 1 \end{bmatrix}.$$

Par la suite, on construit K graphes avec \mathbf{Y} et \mathbf{r} . Notons que la dimension de \mathbf{Y} est maintenant $d \times L$, et que dans toutes les étapes de reconstruction qui suivent, nous travaillons avec les L marqueurs. La reconstruction du graphe est très simple: à chaque pas, les mêmes étapes se répètent (le processus est markovien). Il faut commencer par regarder tous les événements possibles: coalescence de séquences du même type (*PossCoaId*), coalescence de séquences différentes (*PossCoaDi*), mutation (*PossMut*), et recombinaison (*PossRecom*).

PossCoaId(): Événement de coalescence pour des séquences identiques C_i . Une coalescence correspond à l'événement où deux séquences trouvent un ancêtre commun. Un tel événement est possible pour la séquence i si cette séquence a une multiplicité d'au moins deux ($n_i \geq 2$).

PossCoaDi(): Événement de coalescence de séquences différentes C_{ij}^k . Cet événement est possible si deux séquences i et j ne possèdent pas de mutations incompatibles: aucun locus m ($m = 1, \dots, L$) ne doit satisfaire à l'une des

conditions suivantes:

$$\begin{cases} Y_{im} = 1 & \text{et } Y_{jm} = 0 \\ Y_{im} = 0 & \text{et } Y_{jm} = 1. \end{cases}$$

où Y_{im} dénote l'allèle du marqueur m de l'haplotype i . Si un tel événement est possible, il nous faut calculer la nouvelle séquence k résultante de l'événement: pour $m = 1, \dots, L$,

$$\begin{cases} Y_{km} = 0, & \text{si } Y_{im} = 0 \text{ ou } Y_{jm} = 0, \\ Y_{km} = 1, & \text{si } Y_{im} = 1 \text{ ou } Y_{jm} = 1, \\ Y_{km} = \bullet, & \text{si } Y_{im} = \bullet \text{ et } Y_{jm} = \bullet, \end{cases}$$

L'étape suivante consiste alors à vérifier si cette nouvelle séquence k existe déjà, ou s'il faut la créer. La probabilité de l'événement dépend alors de la multiplicité de k , δ_{ik} et δ_{jk} .

PossMut(): Événement de mutation $M_i^j(m)$. Une mutation est possible à chacun des L marqueurs, si une seule séquence de multiplicité 1 possède une mutation à ce locus (une valeur 1). Dû au codage des données, il suffit de lire la matrice \mathbf{Y} en colonne: pour avoir événement de mutation, il faut que la somme en colonne de \mathbf{Y} , en tenant compte des multiplicités, soit exactement 1. Autrement dit, une mutation est possible au marqueur m si et seulement si:

$$\sum_{i=1}^d n_i Y_{im} = 1.$$

Si une mutation est possible, il faut alors chercher dans les différentes séquences quelle est celle qui possède un site mutant à ce locus (séquence i). L'étape suivante consiste alors à construire la séquence j en faisant muter le locus identifié de la séquence i , et à vérifier si la nouvelle séquence j existe déjà, afin de connaître sa multiplicité dont dépend la probabilité de l'événement.

PossRecom(): Événement de recombinaison $R_i^{jk}(p)$. Un événement de recombinaison est possible dans chacun des intervalles ($p = 1, \dots, L-1$), pour chacune des séquences i , si l'intervalle en question est ancestral: cette situation se

présente s'il est inclus entre γ_i et κ_i , que l'on aura déjà calculés. Pour chaque événement de recombinaison possible, il faut créer les séquences parentales j et k , et vérifier si elles existent afin d'obtenir leur multiplicité pour le calcul de la probabilité de l'événement.

Une fois que l'on a évalué tous les événements possibles et que l'on a rassemblé les informations indispensables aux calculs des probabilités $(\alpha_\tau, \beta_\tau)$, on peut calculer ces dernières. Un événement est alors choisi au hasard, proportionnellement à son poids.

Pour passer à l'étape suivante du graphe, il ne reste qu'à ajuster les données et la valeur de certains paramètres. Si une coalescence de séquences de type identique se produit (C_i), il faut ajuster la multiplicité de la séquence i : $n_{i-} = 1$ (notation qui signifie que n_i à l'étape $\tau + 1$ est égal à n_i à l'étape τ moins 1), et enlever la séquence i de \mathbf{Y} si n_i devient nul. Ensuite, s'il s'agit d'un événement de coalescence de séquences différentes (C_{ij}^k), on ajoute la séquence k à \mathbf{Y} si elle n'existe pas encore. Si k existe déjà, on modifie sa multiplicité, tel que $n_{k+} = 1$. Dans tous les cas, on diminue la multiplicité des séquences i et j : $n_{i-} = 1$ et $n_{j-} = 1$. Si n_i ou n_j devient nul, il faut alors enlever la séquence i ou j de \mathbf{Y} . Si l'événement choisi est une mutation ($M_i^j(m)$), on ajoute la séquence j à \mathbf{Y} si elle n'existe pas encore. Si la séquence j existe déjà, on modifie sa multiplicité qui devient $n_{j+} = 1$. Dans tous les cas, on diminue la multiplicité de la séquence i , $n_{i-} = 1$. Si n_i devient nul, il faut alors enlever la séquence i de \mathbf{Y} . Si l'événement choisi est une recombinaison (R_i^{jk}), il faut créer les deux nouvelles séquences et vérifier si elles existaient auparavant. Pour chacune de ces nouvelles séquences, si elles n'existaient pas, il faut les rajouter à la matrice \mathbf{Y} , sinon il suffit d'ajuster la multiplicité de la séquence. Quel que soit l'événement choisi, il faudra recalculer, ou mettre à jour, selon les paramètres, les vecteurs $\boldsymbol{\gamma}$, $\boldsymbol{\kappa}$, \mathbf{a} , \mathbf{b} , \mathbf{n} (qui contiennent de l'information pour chaque type de séquence: par exemple, le vecteur $\boldsymbol{\gamma}$ est de dimension d et contient la valeur de γ_i pour chaque séquence i) et les paramètres α , β .

Une des options concerne le nombre de points pour lesquels la vraisemblance est évaluée ($NbCan$). Si x est la valeur de la variable $NbCan$, le programme répartit les x candidats sur la longueur totale de la séquence, de telle façon que l'on ait x_p points dans l'intervalle p (proportionnellement à la longueur de l'intervalle), et que le nombre total de points soit x . Une autre option s'offre également à l'utilisateur afin de lui permettre de préciser le nombre de points par intervalle; celle-ci permet, par exemple, de ne pas évaluer la vraisemblance dans certains intervalles, tout en utilisant l'ensemble de la séquence pour construire les graphes. Le temps de calcul grandit légèrement en fonction du nombre de points pour lesquels la vraisemblance est évaluée. Cependant, plus le nombre de points est élevé, plus la courbe de vraisemblance est précise. La figure 4.2.2 illustre cette différence. Les deux calculs pour (a) et (b) sont identiques (même racine aléatoire) mais le nombre de points pour lesquels la vraisemblance est évaluée est de 50 dans la première analyse, et de 200 dans la seconde. Pour cet exemple (A), alors qu'il faut 6 heures pour calculer la vraisemblance avec 1M d'itérations avec 50 valeurs candidates, il faut 7 heures et 12 minutes avec 200 valeurs candidates pour le même nombre d'itérations.

La courbe de vraisemblance est habituellement représentée sur toute la longueur de la séquence, mais elle n'est en fait pas définie entre le dernier point d'un

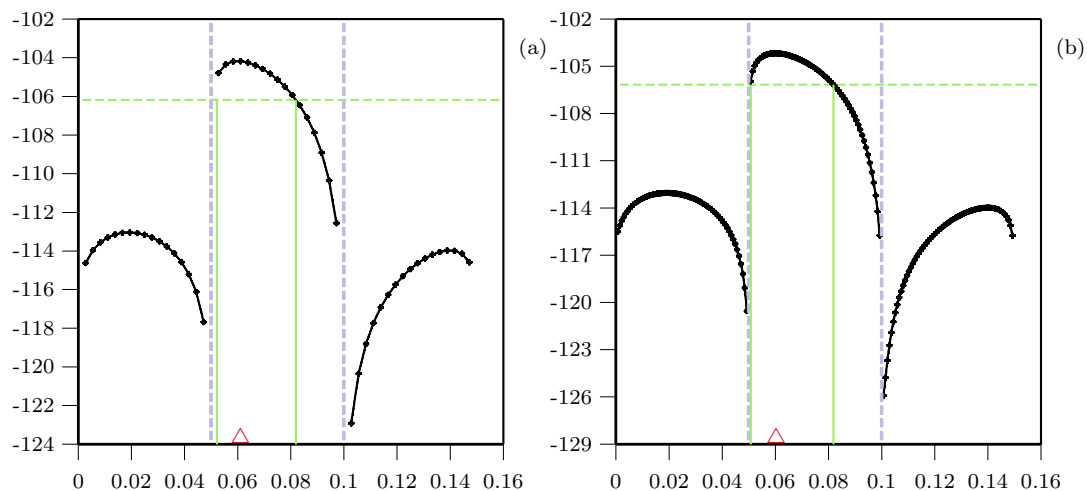


Figure 4.2.2. Illustration de l'effet du nombre de points pour lesquels la vraisemblance est calculée: (a) $NbCan$ est 50, (b) $NbCan$ est 200.

intervalle et le premier point de l'intervalle suivant, comme l'illustre la figure 4.2.2: aux deux séparations entre marqueurs (lignes en pointillé verticales à $x = 0.05$ et $x = 0.10$), la vraisemblance n'est pas représentée.

4.3. Exemple de reconstruction d'un ARG

Afin d'illustrer l'algorithme ainsi que la méthodologie, nous offrons ici un exemple simple, mais assez détaillé pour comprendre le niveau de complexité de ceux-ci. Les données sont:

$$\mathbf{Y}' = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{T} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}, \quad \mathbf{n} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \quad \mathbf{r}' = \begin{bmatrix} 0.02 \\ 0.06 \end{bmatrix}$$

Supposons que nous évaluons la vraisemblance sur le second intervalle. Nous noterons les vecteurs matrices impliqués en fonction de l'étape en cours (τ), par exemple \mathbf{Y}_0 la matrice \mathbf{Y} à l'étape 0, de même que γ_1 pour le vecteur γ à l'étape 1. De plus, nous noterons E_τ l'ensemble des événements possibles, et P_τ leur probabilité associée. Les événements sont toujours fonction des types de séquence, et ceux-ci sont indiqués à gauche de \mathbf{Y} . Pour faciliter la compréhension, une lettre unique identifie chaque séquence; cette lettre qui est reportée près de la dite séquence sur le graphe de la figure 4.3.1. Nous utilisons ici la méthode approximative pour les recombinaisons afin de diminuer le nombre d'événements possibles, et ainsi alléger la présentation. Pour la même raison, on peut noter que le second locus n'est pas polymorphique (ce qui le rend moins informatif), mais nous traitons moins d'événements ainsi.

Les notations que nous utiliserons sont celles que nous avons utilisées dans le chapitre 3: C_i pour une coalescence de deux séquences de type i , C_{ij}^k pour une coalescence des séquences i et j dans une séquence k , $M_i^j(m)$ pour une mutation d'une séquence i au locus m vers une séquence j , et $R(p)$ pour une recombinaison dans l'intervalle p . Comme la méthode approximative pour les recombinaisons

est utilisée, nous ne considérons que les intervalles de recombinaison, c'est-à-dire qu'il y a autant d'événements de recombinaison que d'intervalles; si un événement de recombinaison est choisi, une séquence est alors choisie au hasard parmi les séquences où une recombinaison est ancestrale. Quand la séquence résultante d'un événement est une nouvelle séquence (dont le type n'existe pas à ce moment-là), le numéro de la séquence est -1 ; ainsi, à l'étape 0 par exemple, un des événements possibles est une mutation de la séquence 3 au locus 4 dans une nouvelle séquence dont le type n'est pas présent dans l'échantillon à $\tau = 0$, cet événement est alors noté $M_3^{-1}(4)$.

Avec les données ci-dessus, comme $p = 2$, on a $\mathbf{r} = [0.02, 0.03, 0.03]$. La construction du graphe commence à l'étape zéro. On a alors:

$$n_0 = 4, \quad \alpha_0 = 1, \quad \beta_0 = 1,$$

$$\boxed{\tau = 0} \quad \mathbf{Y}_0 = \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \end{matrix} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{matrix} a \\ b \\ c \\ d \end{matrix}, \quad \mathbf{n}_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \quad \boldsymbol{\gamma}_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \quad \boldsymbol{\kappa}_0 = \begin{bmatrix} 3 \\ 3 \\ 3 \\ 3 \end{bmatrix},$$

$$E_0 = \{M_3^{-1}(4), \quad R(1), \quad R(2), \quad R(3)\},$$

$$P_0 = \{0.556, \quad 0.110, \quad 0.167, \quad 0.167\}.$$

L'événement choisi est $M_3^{-1}(4)(M_d^e(4))$.

$$n_1 = 4, \quad \alpha_0 = 1, \quad \beta_0 = 1,$$

$$\boxed{\tau = 1} \quad \mathbf{Y}_1 = \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \end{matrix} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{matrix} a \\ b \\ c \\ e \end{matrix}, \quad \mathbf{n}_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \quad \boldsymbol{\gamma}_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \quad \boldsymbol{\kappa}_1 = \begin{bmatrix} 3 \\ 3 \\ 3 \\ 3 \end{bmatrix},$$

$$E_1 = \{R(1), \quad R(2), \quad R(3)\},$$

$$P_1 = \{0.25, \quad 0.375, \quad 0.375\}.$$

L'événement choisi est $R(2)$, et la séquence choisie au hasard pour recombiner est la séquence 2 (c), d'où on prend $R_c^{fg}(2)$.

$$n_2 = 5, \quad \alpha_2 = 0.8, \quad \beta_2 = 0.725,$$

$$\boxed{\tau = 2} \quad \mathbf{Y}_2 = \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & \bullet & \bullet \\ \bullet & \bullet & 1 & 0 \end{bmatrix} \begin{array}{c} a \\ b \\ e \\ f \\ g \end{array}, \quad \mathbf{n}_2 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \quad \boldsymbol{\gamma}_2 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 3 \end{bmatrix}, \quad \boldsymbol{\kappa}_2 = \begin{bmatrix} 3 \\ 3 \\ 3 \\ 1 \\ 3 \end{bmatrix},$$

$$E_2 = \{R(1), R(2), R(3), C_{1,3}^1, C_{2,4}^2, C_{3,4}^{-1}\},$$

$$P_2 = \{0.238, 0.269, 0.358, 0.045, 0.045, 0.045\}.$$

L'événement choisi est $R(2)$. La séquence 1 (b) est alors choisie, et on a $R_b^{i,h}$.

$$n_3 = 6, \quad \alpha_3 = 0.667, \quad \beta_2 = 0.542,$$

$$\boxed{\tau = 3} \quad \mathbf{Y}_3 = \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & \bullet & \bullet \\ \bullet & \bullet & 1 & 0 \\ \bullet & \bullet & 0 & 0 \end{bmatrix} \begin{array}{c} a \\ e \\ f/h \\ g \\ i \end{array}, \quad \mathbf{n}_3 = \begin{bmatrix} 1 \\ 1 \\ 2 \\ 1 \\ 1 \end{bmatrix}, \quad \boldsymbol{\gamma}_3 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 3 \\ 3 \end{bmatrix}, \quad \boldsymbol{\kappa}_3 = \begin{bmatrix} 3 \\ 3 \\ 1 \\ 3 \\ 3 \end{bmatrix},$$

$$E_3 = \{R(1), R(2), R(3), C_{1,3}^1, C_{2,3}^{-1}, C_2, C_{0,4}^0, C_{2,4}^{-1}\},$$

$$P_3 = \{0.179, 0.179, 0.359, 0.063, 0.063, 0.031, 0.063, 0.063\}.$$

L'événement choisi est $C_{1,3}^1$, c'est-à-dire C_{eg}^j .

$$n_4 = 5, \quad \alpha_4 = 0.7, \quad \beta_4 = 0.575,$$

$$\boxed{\tau = 4} \quad \mathbf{Y}_4 = \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & \bullet & \bullet \\ \bullet & \bullet & 0 & 0 \end{bmatrix} \begin{array}{c} a \\ j \\ f/h \\ i \end{array}, \quad \mathbf{n}_4 = \begin{bmatrix} 1 \\ 1 \\ 2 \\ 1 \end{bmatrix}, \quad \boldsymbol{\gamma}_4 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 3 \end{bmatrix}, \quad \boldsymbol{\kappa}_4 = \begin{bmatrix} 3 \\ 3 \\ 1 \\ 3 \end{bmatrix},$$

$$E_4 = \{R(1), R(2), R(3), C_2, C_{0,3}^0, C_{2,3}^{-1}, M_1(3)\},$$

$$P_4 = \{0.050, 0.050, 0.075, 0.006, 0.012, 0.012, 0.795\}.$$

L'événement choisi est $M_1(3)$, c'est-à-dire $M_j^k(3)$.

$$n_5 = 5, \quad \alpha_5 = 0.7, \quad \beta_5 = 0.575,$$

$$\boxed{\tau = 5} \quad \mathbf{Y}_5 = \begin{array}{c} 0 \\ 1 \\ 2 \end{array} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & \bullet & \bullet \\ \bullet & \bullet & 0 & 0 \end{bmatrix} \begin{array}{c} a/k \\ f/h \\ i \end{array}, \quad \mathbf{n}_5 = \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix}, \quad \boldsymbol{\gamma}_5 = \begin{bmatrix} 1 \\ 1 \\ 3 \end{bmatrix}, \quad \boldsymbol{\kappa}_5 = \begin{bmatrix} 3 \\ 1 \\ 3 \end{bmatrix},$$

$$E_5 = \{R(1), R(2), R(3), C_1, C_{0,2}^0, C_{1,2}^{-1}, C_0\},$$

$$P_5 = \{0.211, 0.158, 0.316, 0.039, 0.158, 0.079, 0.039\}.$$

L'événement choisi est $C_{0,2}^0$, c'est-à-dire C_{ai}^l

$$n_6 = 4, \quad \alpha_6 = 0.75, \quad \beta_6 = 0.625,$$

$$\boxed{\tau = 6} \quad \mathbf{Y}_6 = \begin{array}{c} 0 \\ 1 \end{array} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & \bullet & \bullet \end{bmatrix} \begin{array}{l} l/k \\ f/h \end{array}, \quad \mathbf{n}_6 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \quad \boldsymbol{\gamma}_6 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \boldsymbol{\kappa}_6 = \begin{bmatrix} 3 \\ 1 \end{bmatrix},$$

$$\begin{aligned} E_6 &= \{R(1), R(2), R(3), C_1, C_0\}, \\ P_6 &= \{0.364, 0.273, 0.273, 0.045, 0.045\}. \end{aligned}$$

L'événement choisi est C_1 , c'est-à-dire C_{fh}^m .

$$n_7 = 3, \quad \alpha_7 = 0.75, \quad \beta_7 = 0.625,$$

$$\boxed{\tau = 7} \quad \mathbf{Y}_7 = \begin{array}{c} 0 \\ 1 \end{array} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & \bullet & \bullet \end{bmatrix} \begin{array}{l} l/k \\ m \end{array}, \quad \mathbf{n}_7 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \quad \boldsymbol{\gamma}_7 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \boldsymbol{\kappa}_7 = \begin{bmatrix} 3 \\ 1 \end{bmatrix},$$

$$\begin{aligned} E_7 &= \{R(1), R(2), R(3), C_0, M_1^{-1}(1)\}, \\ P_7 &= \{0.095, 0.071, 0.071, 0.007, 0.757\}. \end{aligned}$$

L'événement choisi est $M_1^{-1}(1)$, c'est-à-dire $M_m^n(1)$.

$$n_8 = 2, \quad \alpha_8 = 0.833, \quad \beta_8 = 0.75,$$

$$\boxed{\tau = 8} \quad \mathbf{Y}_8 = \begin{array}{c} 0 \\ 1 \end{array} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & \bullet & \bullet \end{bmatrix} \begin{array}{l} l/k \\ n \end{array}, \quad \mathbf{n}_8 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \quad \boldsymbol{\gamma}_8 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \boldsymbol{\kappa}_8 = \begin{bmatrix} 3 \\ 1 \end{bmatrix},$$

$$\begin{aligned} E_8 &= \{R(1), R(2), R(3), C_0, C_{0,1}^0\}, \\ P_8 &= \{0.348, 0.261, 0.261, 0.026, 0.104\}. \end{aligned}$$

L'événement choisi est $C_{0,1}^0$, c'est-à-dire C_{nk}^o

$$n_9 = 3, \quad \alpha_9 = 1, \quad \beta_9 = 1,$$

$$\boxed{\tau = 9} \quad \mathbf{Y}_9 = 0 \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix} l/o, \quad \mathbf{n}_9 = [2], \quad \boldsymbol{\gamma}_9 = [1], \quad \boldsymbol{\kappa}_9 = [3],$$

$$\begin{aligned} E_9 &= \{R(1), R(2), R(3), C_0\}, \\ P_9 &= \{0.245, 0.368, 0.368, 0.018\}. \end{aligned}$$

L'événement choisi est C_0 , c'est-à-dire C_{lo}^p

$$n_{10} = 1, \quad \alpha_{10} = 1, \quad \beta_{10} = 1,$$

$$\boxed{\tau = 10} \quad \mathbf{Y}_{10} = 0 \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix} p, \quad \mathbf{n}_{10} = [1], \quad \boldsymbol{\gamma}_{10} = [1], \quad \boldsymbol{\kappa}_{10} = [3],$$

La construction du graphe est alors achevée car une séquence unique avec des 0 partout, l'ancêtre commun des quatre séquences, a été trouvé. La figure 4.3.1 illustre le graphe construit dans cet exemple.

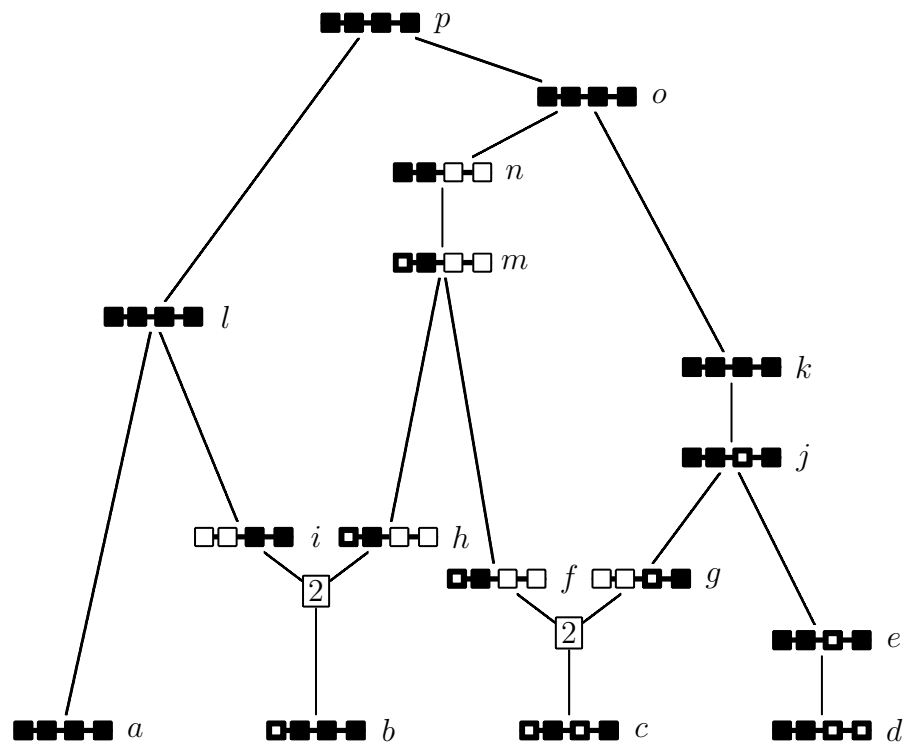


Figure 4.3.1. Le graphe de recombinaison ancestral correspondant aux calculs effectués dans la section 4.3.

4.4. Amélioration de l'algorithme précédent

Le programme correspondant à l'algorithme précédent nécessite beaucoup de temps de calcul, surtout pour des exemples réels. On parle ici de plusieurs semaines de calcul. Il est donc primordial d'améliorer la performance de l'algorithme. Nous présentons ici une version plus performante la plupart du temps. Commençons par faire une évaluation sommaire des calculs que nous faisons présentement, afin de voir comment on peut gagner du temps de calcul.

Les calculs pour évaluer si une coalescence de séquences identiques est possible sont simples, puisque seule la multiplicité des séquences est un critère: il suffit de consulter le vecteur contenant la multiplicité des séquences. Pour évaluer si des coalescences de séquences de différents types (C_{ij}^k) sont possibles, il faut vérifier si une coalescence est possible entre les d_τ séquences (où d_τ est le nombre de séquences différentes à l'étape τ), donc effectuer $d_\tau(d_\tau - 1)/2$ vérifications. Afin de savoir si une coalescence est possible pour une paire de séquences, il n'y a pas d'autres solutions que de comparer un à un tous les marqueurs, c'est-à-dire faire L comparaisons au maximum. En effet, quand on compare si deux séquences sont identiques, le nombre d'opérations est au maximum L , car si l'on constate que le second marqueur, par exemple, est différent, on sait que les séquences sont différentes, quelle que soit la suite des séquences. Le nombre d'opérations associé à la vérification des événements de coalescence de séquences de types différents est donc $L \times d_\tau(d_\tau - 1)/2$. Par exemple, pour la première étape dans le cas de l'exemple D, on a 105 coalescences à vérifier ($d_0=15$), et puisque $L = 6$, le nombre d'opérations nécessaires est donc de 630. Le même calcul dans le cas de l'exemple E nous donne 42 090 ($d_\tau = 61$ et $L = 23$). Si c coalescences sont possibles, il faut alors déduire la nouvelle séquence pour chacun des c événements, et voir si elle existe déjà ou non: nous ajoutons ainsi $c \times d_\tau \times L$ opérations (au maximum). Évidemment, les ordinateurs modernes calculent rapidement, mais il faut prendre en considération qu'un seul graphe peut nécessiter de quelques dizaines à quelques milliers d'étapes pour sa construction, et que des centaines de milliers, voir des millions de graphes sont nécessaires afin d'évaluer la vraisemblance.

Les calculs nécessaires pour évaluer les événements de mutation possibles demandent moins de temps. En effet, une mutation n'est possible que si le site mutant est unique. Un vecteur dénoté par V , de dimension L , contenant la somme (pondérée par les multiplicités des séquences) en colonne de Y nous informe si une mutation est possible:

$$V_l = \sum_{i=1}^d n_i Y_{il}, \quad l = 1, \dots, L.$$

Si un élément du vecteur V est 1, alors une mutation est possible; il suffit de chercher quelle séquence possède la mutation, ce qui représente $d_\tau \times L$ opérations au maximum.

À moins que l'on n'utilise la méthode approximative pour les recombinaisons (comme le présente la section 3.11), chaque séquence peut recombiner à chaque intervalle si celui-ci est ancestral. Pour chaque recombinaison, il est nécessaire de créer temporairement les séquences parentales, afin de trouver leur multiplicité. Une approximation est que, pour chacune des d_τ séquences, $L - 1$ recombinaisons sont possibles, et à chacune d'elle, $2L$ opérations sont nécessaires pour vérifier si les parents existent. Donc un nombre d'opérations total de l'ordre de $d_\tau \times (L - 1) \times 2L$ qui donne pour la première étape de l'exemple D, 900 opérations, et 61 732 pour celle de l'exemple E.

Le nombre d'opérations nécessaire pour évaluer tous les événements possibles et leur probabilité étant considérable, une solution a été cherchée afin de diminuer le temps de calcul. Comme nous l'avons vu, à chaque étape de la construction du graphe nous recalculons tous les événements possibles. Or, entre deux étapes successives de la construction d'un graphe, la liste des événements possibles est presque identique. Par exemple, si à l'étape τ , l'événement $C_{2,6}^5$ est possible, il est probable qu'il le soit encore à l'étape $\tau + 1$, car parmi tous les événements possibles, un seul est choisi, et celui-ci ne modifie que trois séquences au maximum (dans le cas d'une recombinaison ou d'une coalescence). Si nous prenons la liste des événements possibles aux étapes 4 et 5 de l'exemple de la section 4.3, sept événements sont possibles à chacune de ces deux étapes, mais un seul est différent. De façon générale, parmi tous les événements possibles à l'étape τ , la plupart sont encore possibles à l'étape $\tau + 1$. Notons par contre que leur probabilité a changé. Nous modifions donc l'algorithme pour ne pas avoir à recalculer systématiquement tous les événements possibles à chaque étape. Pour cela, nous modifions la liste des événements d'une étape à l'autre. Il suffit de prévoir toutes les conséquences qu'un événement a sur la liste des événements possibles.

À la première étape de la reconstruction du graphe, il faut commencer par établir une liste des événements possibles en utilisant la méthode précédente. Une fois l'événement choisi, il faut mettre à jour les paramètres et données avec le moins d'opérations possibles. Dans le cas d'une coalescence de séquences de type identique C_i , il suffit de mettre à jour la valeur de n_i (qui diminue de 1). Si n'_i est 1 (nous noterons n'_i la nouvelle multiplicité de la séquence i après qu'un événement ait lieu), il faut aussi retirer l'événement en question de la liste des événements possibles, puisqu'une coalescence identique de cette séquence devient impossible, et vérifier si de nouvelles mutations sont possibles. En effet, le fait de changer la multiplicité de la séquence peut créer de nouveaux événements de mutation.

Si l'événement choisi est une coalescence de séquences différentes C_{ij}^k , il faut générer la séquence k , à moins qu'elle n'existe déjà; si $n'_i = 0$ ($n'_j = 0$), il faut enlever (ou modifier) tous les événements impliquant la séquence i (j); il est en effet possible qu'il existe un autre événement de coalescence impliquant une des séquences i ou j . Si $n_i = 1$ ($n_j = 1$), il faut supprimer les événements de coalescence C_i et C_j qui étaient dans la liste des événements possibles. Si une nouvelle séquence k est créée, c'est-à-dire si $n'_k = 1$, il faut vérifier si une coalescence est possible entre la séquence k et les $d_\tau - 1$ autres séquences. Il faut également vérifier si la nouvelle séquence k est impliquée dans d'autres événements: par exemple, s'il existe un événement $C_{i'j'}^k$ mais que la séquence k n'existait pas encore, alors il faut recalculer la probabilité de l'événement. Si $n'_k = 2$, il faut ajouter un événement de coalescence C_k . Enfin, il faut vérifier si l'événement en question a des implications sur les événements de mutation: de nouvelles mutations peuvent être possibles.

Dans le cas d'une mutation M_i^j , il faut retirer les événements où la séquence i est impliquée et enlever la séquence i du vecteur des séquences, puisque sa multiplicité est maintenant nulle. Si $n'_j = 1$, c'est-à-dire que la séquence j est une nouvelle séquence, il faut vérifier si de nouvelles coalescences de séquences

de types différents sont possibles, et si cette nouvelle séquence est impliquée dans d'autres événements. Finalement, si $n'_j = 2$, il faut ajouter l'événement C_j à la liste des événements possibles.

Pour un événement de recombinaison R_i^{jk} , il faut effacer la séquence i et tous les événements qui y sont reliés si $n'_i = 0$. Si $n'_i = 1$, il faut enlever l'événement C_i de la liste des événements. Si $n'_j = 1$ ($n'_k = 1$), il faut vérifier si de nouvelles coalescences de séquences différentes impliquant la séquence j (k) sont possibles, et si parmi les événements possibles, certains impliquent les nouvelles séquences j et k . Enfin, si $n_j = 2$ ($n'_k = 2$) il faut ajouter l'événement C_i (C_j).

Ce nouvel algorithme donne des résultats concluants: il est plus rapide, de l'ordre d'une fois et demi à deux fois, que le précédent.

4.5. Amélioration supplémentaire de l'algorithme

Le temps d'exécution du programme qui implante la méthode est un facteur important. Nous avons constaté que les calculs nécessaires requis pour passer en revue les événements de coalescence de séquences de types différents étaient coûteux. L'identification des séquences résultant de tels événements est également une opération qui nécessite beaucoup de calculs. Dans certains cas, une nouvelle amélioration, basée sur la précédente (section 4.4) est possible. Celle-ci consiste à utiliser une série de tables contenant les informations sur tous les événements possibles avant de commencer à construire des graphes, dans le but de réduire le nombre d'instructions nécessaires au moment de la construction. Un exemple de ces tables est illustré à la figure 4.5.1, pour le cas $L = 3$.

La première étape consiste à construire un tableau (THap) de tous les haplotypes possibles: chaque marqueur ayant trois états possibles, cette matrice a dimension $(3^L - 1) \times L$. Ensuite, un tableau (Ta) contenant le nombre de sites ancestraux pour chaque séquence est créé, puis un autre pour la longueur où une recombinaison ancestrale est possible (Tb), ces tableaux ayant pour dimension $(3^L - 1) \times 1$. Puis, un tableau contenant toutes les mutations possibles (TMu), de

dimension $(3^L - 1) \times L$; il identifie la séquence résultant d'un événement de mutation. Ainsi, pour connaître la séquence résultant d'une mutation au marqueur 3 de la séquence 7, il suffit de consulter l'entrée TMu [7] [2] pour savoir que l'on obtient une séquence 6 (pour $L = 3$, tel qu'illustré à la figure 4.5.1. Un tableau de dimension $(3^{L-1}) \times 2$ contient les paramètres γ et κ de chaque séquence (TAnc); par exemple, une séquence 5 est admissible pour les recombinaisons entre les marqueurs 1 et 2 (TAnc [5] [0] et TAnc [5] [1]). Deux tableaux identifient les séquences de droite et de gauche résultant des événements de recombinaison, chacun de

0	0 0 0					-	-	-	1 3
1	0 0 1	-				-	-	0	1 3
2	0 0 -	0 1				-	-	-	1 2
3	0 1 0	-	-	-		-	0	-	1 3
4	0 1 1	-	-	-	-	-	1	3	1 3
5	0 1 -	-	-	-	3 4	-	2	-	1 2
6	0 - 0	0 - 0	3 - 3			-	-	-	1 3
7	0 - 1	- 1 1	- 4 4 -			-	-	6	1 3
8	0 - -	0 1 2 3 4 5 6 7				-	-	-	1 1
9	1 0 0	-	-	-	-	-	0	-	1 3
10	1 0 1	-	-	-	-	-	1	- 9	1 3
11	1 0 -	-	-	-	-	-	2	-	1 2
12	1 1 0	-	-	-	-	-	3	9 -	1 3
13	1 1 1	-	-	-	-	-	4	10 12	1 3
14	1 1 -	-	-	-	-	- 12 13	5	11 -	1 2
15	1 - 0	-	-	-	-	- 9 - 9 12 - 12	6	-	1 3
16	1 - 1	-	-	-	-	- 10 10 - 13 13 -	7	- 15	1 3
17	1 - -	-	-	-	-	- 9 10 11 12 13 14 15 16	8	-	1 1
18	- 0 0	0 - 0 -	- 0 - 0 9 - 9 -	-	-	- 9 - 9	-	-	2 3
19	- 0 1	- 1 1 -	- 1 1 - 10 10 -	-	-	- 10 10 -	-	- 18	2 3
20	- 0 -	0 1 2 -	- 0 1 2 9 10 11 -	-	-	- 9 10 11 18 19	-	-	2 2
21	- 1 0	-	- 3 - 3 3 - 3 -	-	-	- 12 - 12 12 - 12 - - -	-	- 18	2 3
22	- 1 1	-	- 4 4 - 4 4 -	-	-	- 13 13 - 13 13 - - - -	-	- 19 21	2 3
23	- 1 -	-	- 3 4 5 3 4 5 -	-	-	- 12 13 14 12 13 14 - - - 21 22	-	- 20 -	2 2
24	- - 0	0 - 0 3 -	3 6 - 6 9 - 9 12 -	12 15 - 15 18 - 18 21 -	21	-	-	-	3 3
25	- - 1	- 1 1 - 4 4 -	7 7 - 10 10 -	13 13 - 16 16 - 19 19 -	22 22 -	-	-	- 24	3 3

Id THap TCoa TMu TAnc

Figure 4.5.1. Illustration des matrices pour $L = 3$. Sont présentés de gauche à droite le numéro unique d'identification de chacune des 25 séquences possibles (Id), les 25 séquences elles-mêmes (THap), la matrice d'information sur les coalescences (TCoa), la matrice d'information sur les mutations (TMu), et la matrice d'information du matériel ancestral pour les recombinaisons (TAnc: γ et κ). Sont omis de cette figure pour des raisons d'espace, mais calculés cependant, les vecteurs contenant l'information sur le contenu ancestral de chaque séquence, a et b (Ta et Tb), et les matrices contenant l'information sur les parents des séquences recombinantes à gauche (TReL) et à droite (TReR). Le symbole "-" indique un marqueur non ancestral pour THap, et un événement impossible pour TCoa et TMu.

dimension $(3^L - 1) \times (L - 1)$. Un tableau de coalescence identifie les séquences résultant de toutes les coalescences possibles des $(3^L - 1)$ séquences; c'est une matrice triangulaire de $(3^L - 1)$ lignes, et au maximum de $(3^L - 2)$ colonnes; ainsi on peut facilement savoir qu'une coalescence entre une séquence 6 et une séquence 1 est impossible (TCoa[6] [1]), et que le résultat d'une coalescence entre une séquence 6 et une séquence 3 est une séquence 3 (TCoa[6] [3]). Enfin, un vecteur contient la multiplicité de chacune des séquences.

Tous ces calculs préliminaires nous permettent d'alléger considérablement le programme, de ne plus vérifier marqueur par marqueur si une coalescence est possible par exemple. Malgré le calcul préliminaire imposant à effectuer, cet algorithme nous permet de diminuer de moitié le temps total d'exécution, ce qui est considérable. Malheureusement, ceci n'est faisable qu'avec de petits ensembles de données. Pour l'exemple D, on a $L = 5$, donc le nombre de séquences possibles est 728, et le nombre de coalescences différentes possibles est $[(3^{L+1} - 2)((3^{L+1} - 1))]/2$, soit 264628. Ces calculs ne prennent cependant que quelques secondes. Pour l'exemple E, où l'on a $L = 23$, le nombre de séquences possibles est alors 2.82×10^{11} ; le calcul devient alors astronomique. Cette modification nous permet également d'éviter d'avoir à gérer les numéros de séquences, puisque l'on a un numéro d'identification unique à chacune des $3^L - 1$ séquences. Dans les versions précédentes, les numéros des séquences changeaient à chaque étape, comme on peut le voir dans l'exemple de la section 4.3. Ainsi la séquence $(1, 0, \bullet, \bullet)$ est la séquence numéro deux à l'étape quatre, mais porte le numéro un à l'étape 5. Il faut alors changer les numéros des séquences dans la liste des événements possibles. Par exemple, la coalescence C_{03}^0 de l'étape 4 réfère au même événement que C_{02}^0 de l'étape 5, seul le numéro de la séquence a changé. Avec la modification proposée dans cette section, nous n'avons plus ce problème.

4.6. Programmation

Un des aspects les plus importants de la méthode est que le programme fasse bien ce que l'on attend de lui. Le nombre de lignes de codes, la grandeur du nombre de simulations faites (des dizaines de millions) font qu'une erreur pourrait se glisser parmi les 4000 lignes de codes. Il est donc important de prendre toutes les précautions pour détecter de tels problèmes.

Évidemment, de multiples vérifications sont effectuées dans le détail pour s'assurer que le programme fait réellement le bon travail. Dans le programme, aucune vérification n'est faite quand l'on essaie d'accéder à une ligne de la matrice Y par exemple; ceci permet de détecter dès les premiers essais en développement des problèmes potentiels, et a aussi le mérite d'être plus rapide. Cela assure que la logique sous-jacente à la programmation est respectée. De plus, des outils de vérification (temporaire) sont incorporés dans le programme pour vérifier tous les paramètres vérifiables; ces outils sont conçus tels qu'ils calculent les paramètres d'une façon différente de celle qui est faite dans le programme. À chaque nouvelle version développée, on fait coexister dans le programme les deux méthodes, et on vérifie que les deux font la même chose.

Le programme fonctionne bien: des simulations durant jusqu'à plusieurs semaines ont été réalisées sans problème. Des problèmes d'instabilité ont cependant été rencontrés sur des stations Linux.

4.7. Aspects numériques

Il existe des problèmes numériques liés au calcul de la vraisemblance. Comme nous l'avons vu, la vraisemblance est estimée par:

$$L(r_T) = E_P \left[\prod_{\tau=0}^{\tau^*-1} f(\mathbf{H}_\tau, \mathbf{H}_{\tau+1}) \right].$$

Habituellement, la fonction f à chaque étape du graphe est plus petite que un, alors que le nombre d'étapes, τ^* , est grand, particulièrement si le nombre et la

longueur des séquences sont grands. Le produit dans l'équation ci-dessus est alors très petit, plus petit en fait que la précision usuelle que permet un ordinateur. Une approche pour solutionner ce problème consiste à changer temporairement l'échelle du produit, tel que λL , pour un λ fixé, est estimé avec K simulations par:

$$\lambda \frac{1}{K} \sum_{i=1}^K \left[\prod_{\tau=0}^{\tau^*-1} f(\mathbf{H}_\tau, \mathbf{H}_{\tau+1}) \right] = \frac{1}{K} \sum_{i=1}^K \lambda \left[\prod_{\tau=0}^{\tau^*-1} f(\mathbf{H}_\tau, \mathbf{H}_{\tau+1}) \right].$$

En choisissant un λ de l'ordre de 1×10^{300} , le produit est généralement calculable; quand il ne l'est pas, on considère alors que le produit du graphe est nul. Il suffit alors de calculer $\ln(\lambda L) - \ln \lambda$ pour obtenir $\ln(L)$. C'est cette version que nous utiliserons pour la plupart des calculs de cette thèse (sauf pour le modèle microsatellite que nous présentons dans le prochain chapitre). Cette approche fonctionne, mais elle est encore limitée par la précision de l'ordinateur. Elle a par contre l'avantage d'être beaucoup plus rapide que la méthode plus précise qui suit.

Une autre approche consiste à utiliser des fonctions de calcul de grands nombres (*extended numeric range floating point type*). De telles fonctions existent et sont disponibles publiquement. Ainsi, le “*Southwest Biotechnology and Informatics Center*” offre la librairie `EXTNUM` (NBIC, 2001), qui peut être utilisée à cette fin. Un nombre de type `EXTNUM` peut prendre des valeurs (positive ou négative) entre 1×10^{-646456993} et 2×10^{646456992} , et résout donc le problème.

Un autre point important est d'utiliser un bon générateur aléatoire qui doit être rapide et avoir une longue période. Nous avons utilisé l'implantation de Wagner (2001) en `C++` du générateur “Mersenne Twister” (Matsumoto et Nishimura, 1998), qui a une période de $2^{19937} - 1$.

4.8. Interface du programme

Tous les aspects techniques sont décrits dans le manuel d'instruction (appendice B), nous nous contentons d'en donner ici un aperçu. Le programme est conçu pour recevoir des fichiers ASCII (avec fin de ligne UNIX) en entrée. Deux fichiers d'entrée sont nécessaires: un fichier contenant les données (séquences) et un fichier contenant les divers paramètres du modèle. Le programme, qui se nomme `maparg.exe`, est exécuté dans un shell DOS ou UNIX de façon identique, et les options se précisent de la même manière que des commandes UNIX standard:

```
./maparg.exe -p fic_paramètre -o fichier_sortie -K 5M
```

On suppose ici que le fichier `fic_paramètre.par` existe. Le programme va automatiquement créer quatre fichiers:

- (1) `fichier_sortie.txt` qui contient les paramètres utilisés, le nom des fichiers d'entrée, les résultats;
- (2) `fichier_sortie.don` qui contient les données seulement. Ce fichier peut alors être importé directement dans des logiciels appropriés afin d'obtenir des graphiques ;
- (3) `fichier_sortie.plt` fichier texte qui contient les commandes *GnuPlot* pour voir directement le profil de vraisemblance;
- (4) `fichier_sortie.tex` fichier formaté \TeX qui contient les paramètres utilisés, le nom des fichiers d'entrée, et un graphique de la vraisemblance. Le graphique est construit en utilisant la librairie *DraTeX* qui permet de dessiner.

Le seul paramètre que le programme nécessite est le nom du fichier contenant les paramètres de l'analyse; tous les autres peuvent être précisés dans le fichier paramètres.

Chapitre 5

Améliorations et extensions de la méthode

5.1. Adaptation aux marqueurs microsatellites

Nous avons modélisé jusqu'à présent l'évolution de marqueurs binaires ayant de faibles taux de mutation, comme des SNPs (Single Nucleotide Polymorphisms). Nous allons maintenant considérer des marqueurs microsatellites. Comme nous l'avons brièvement vu dans l'introduction, les marqueurs microsatellites sont des marqueurs ayant des taux de mutation plus élevés, de l'ordre de 10^{-3} à 10^{-5} par génération, et indiquent le nombre de répétitions d'un motif d'ADN. Ainsi, la valeur 133 indique, par exemple, que le motif "ATTC" est répété 133 fois à un locus donné. Un autre haplotype pourra avoir la valeur 134, puis un autre 139, par exemple. Au locus considéré, le nombre de valeurs différentes est fini, et il est habituellement de l'ordre d'une dizaine.

La figure 5.1.1 présente une illustration de ce que pourrait être un arbre de recombinaison ancestral dans lequel l'histoire des séquences est suivie à l'aide de marqueurs microsatellites. Pour simplifier la présentation, nous supposons que le nombre de répétitions est entre 0 et dix, centré sur cinq (où 0 correspond au nombre de répétitions minium). Des couleurs sont associées aux nombres de répétitions des microsatellites.

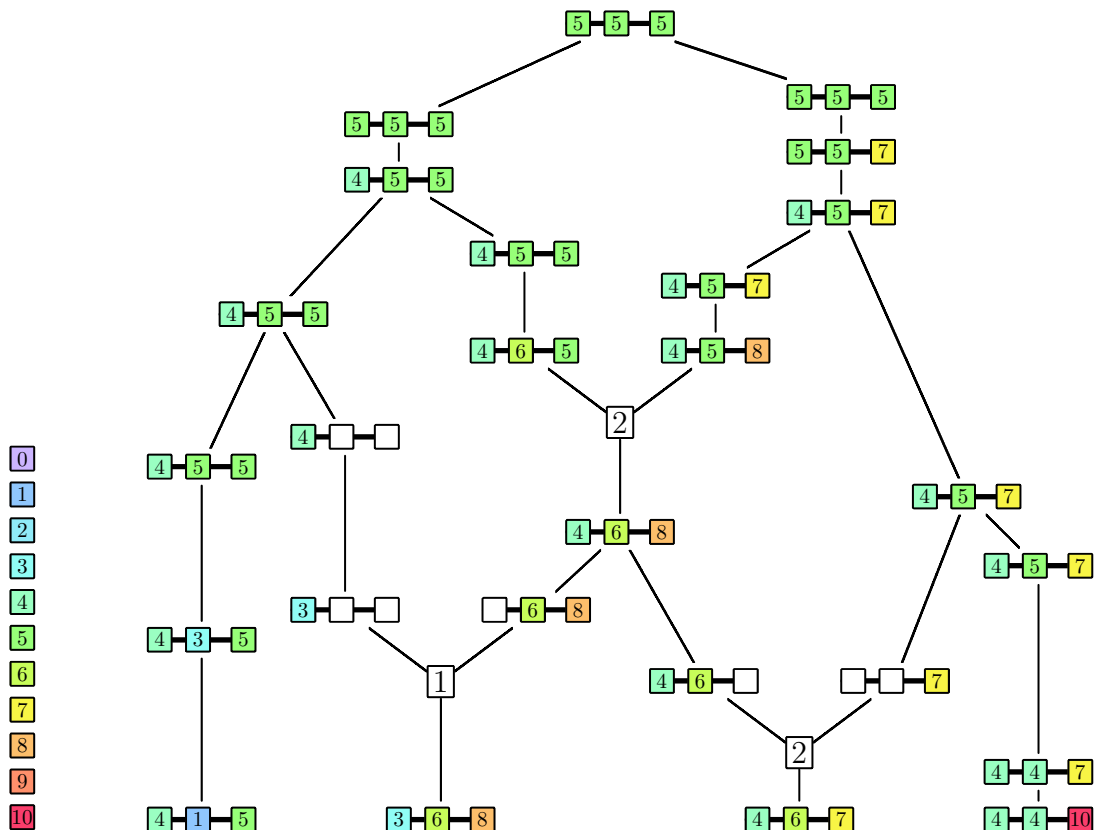


Figure 5.1.1. Illustration d'une généalogie de séquences suivie à l'aide de marqueurs microsatellites. Les nombres de répétitions sont représentés par des couleurs.

Jusqu'à présent, nous avons considéré un modèle où les mutations sont non récurrentes, de telle sorte que chaque marqueur mutant n'est dû qu'à un seul événement de mutation dans l'histoire ancestrale. Cela implique qu'un site mutant ne pourra revenir à son état ancestral que si l'haplotype dont il fait partie a pour multiplicité 1. Ce modèle, tout à fait réaliste si nous travaillons avec des marqueurs de type SNP, a l'avantage de simplifier le modèle. Ainsi, par exem-

ple, comme le montre la figure 5.1.2, les séquences s^1 , s^2 et s^3 ne peuvent pas alors coalescer car elles ont des ensembles de mutations incompatibles. Avec ce modèle, il suffit que l'on ait au moins trois séquences i, j et k de type $(1, 0)$, $(1, 1)$ et $(0, 1)$ pour deux marqueurs quelconques (où 0 représente l'état ancestral, et 1 une mutation) pour qu'une recombinaison entre ces deux sites soit nécessaire. Ainsi, dans la figure 5.1.2, on peut voir que les séquences s^1 et s^2 ne peuvent pas coalescer car la séquence s^2 a une mutation au site 3 que la séquence s^1 ne possède pas; il en est de même pour les coalescences des séquences s^1 et s^3 , et des séquences s^2 et s^3 . D'autre part, aucune mutation n'est possible, ni au site 1, ni au site 3, car le modèle suppose que les mutations ne se produisent qu'une seule fois dans l'histoire. Il faut donc que l'on ait une mutation sur une séquence de multiplicité 1 et que cette mutation ne se retrouve sur aucune autre séquence ancestrale pour qu'un événement de mutation soit possible. Finalement, le seul événement possible est une recombinaison.

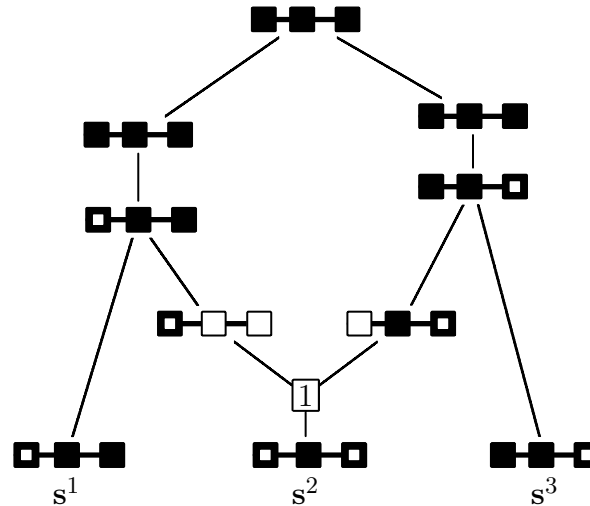


Figure 5.1.2. Illustration d'une généalogie de séquences suivie à l'aide de marqueurs binaires à faible taux de mutation, dans le cadre d'un modèle où les mutations sont non récurrentes.

Maintenant, imaginons une généalogie similaire, mais modélisée à l'aide de marqueurs microsatellites. Une telle généalogie est illustrée dans la partie (a) de la figure 5.1.3. La partie (b) de la figure 5.1.3 montre une généalogie qui n'utilise pas d'événements de recombinaison. Comme on peut le voir, plusieurs

mutations successives sont possibles, et l'on peut relier les trois séquences à l'ancêtre commun sans recombinaison. Or, pour faire de la cartographie, ce sont les événements de recombinaison qui sont informatifs. On peut donc penser que les microsatellites se prêtent moins bien à faire de la cartographie fine que des marqueurs à faible taux de mutation.

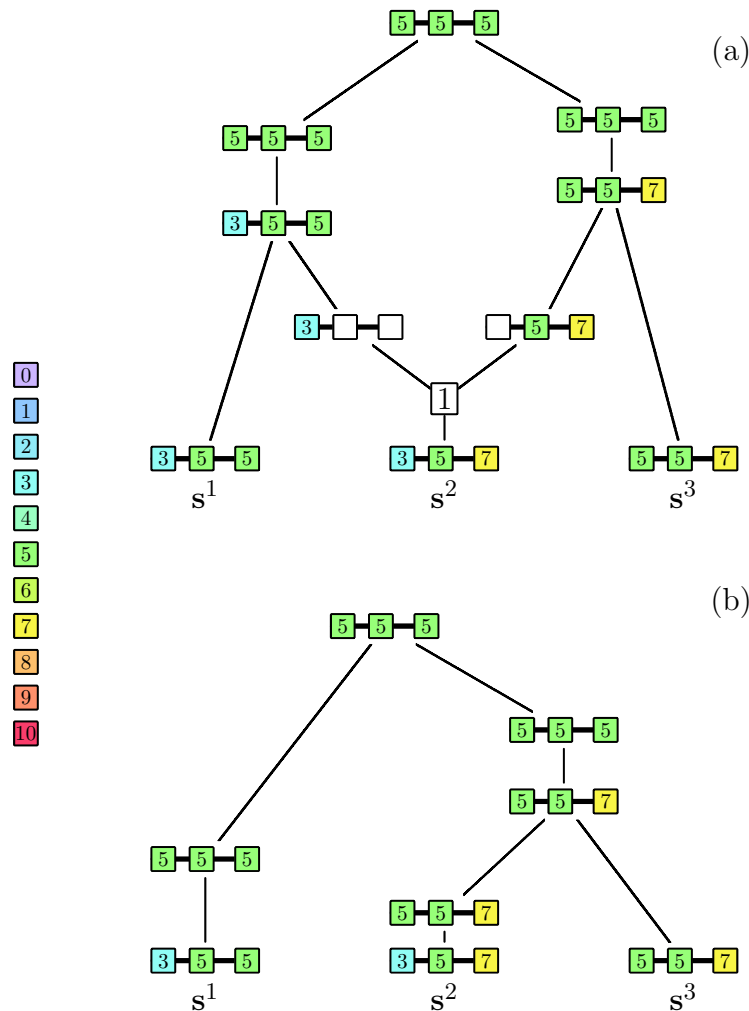


Figure 5.1.3. Illustration d'une généalogie de séquences suivie à l'aide de marqueurs microsatellites: (a) généalogie réelle, (b) généalogie sans recombinaison.

Soit s_m l'allèle du marqueur m ($m = 1, \dots, L$) pour une certaine séquence $\mathbf{s} = (s_1, \dots, s_L)$. Une séquence est par exemple $(3_1, 5_2, 5_3)$ pour la séquence \mathbf{s}^1 de la figure 5.1.3. La probabilité conditionnelle en avant dans le temps, sachant que l'on a un événement de mutation au marqueur m , pour que s_m mute vers $(s+1)_m$

où $\underline{\Delta}_m \leq (s \pm \phi)_m \leq \overline{\Delta}_m$. Nous avons là un modèle de mutation très général qui peut être simplifié. Certaines méthodes vont supposer, par exemple, que $v_m^{-h} = v_m^{+h}$, que $\phi = 1$, ou que $v_m^{\pm h} = v$ quel que soit m , pour $h = 1, \dots, \phi$. La matrice de transition d'un modèle en échelle par exemple est:

$$P_m = \begin{matrix} & \cdots & (s-2)_m & (s-1)_m & s_m & (s+1)_m & (s+2)_m & \cdots \\ \cdots & \left(\begin{array}{ccccccc} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & 0 & v_m & 0 & 0 & 0 & \cdots \\ \cdots & v_m & 0 & v_m & 0 & 0 & \cdots \\ \cdots & 0 & v_m & 0 & v_m & 0 & \cdots \\ \cdots & 0 & 0 & v_m & 0 & v_m & \cdots \\ \cdots & 0 & 0 & 0 & v_m & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{array} \right) & \end{matrix}$$

où $\underline{\Delta}_m \leq (s \pm \phi)_m \leq \overline{\Delta}_m$. Notons que des modèles de substitution incluent la probabilité que \mathbf{s}_m mute vers \mathbf{s}_m cependant, avec le graphe de recombinaison ancestral, il n'est pas pertinent de supposer qu'une telle probabilité n'est pas nulle, puisqu'un tel événement de mutation ne causerait aucun changement dans le graphe. Si des données appropriées sont disponibles, toutes les probabilités de la matrice P_m peuvent être précisées. Toutefois, ces données ne sont habituellement pas connues. Un modèle pour décrire le changement du nombre de répétitions d'ordre supérieur à 1 est le modèle géométrique symétrique, de paramètre p (Pritchard *et al.*, 1999):

$$v_m^{|h|} = \begin{cases} 2^{-1}p(1-p)^{|h|-1}, & \text{pour } |h| \geq 1 \\ 0 & \text{sinon.} \end{cases}$$

Cependant, si le changement dans le nombre de répétitions du microsatellite est au maximum de ϕ , alors on a:

$$v_m^{|h|} = \frac{1}{2}p(1-p)^{|h|-1} \left(\sum_{\substack{m=-\phi \\ m \neq 0}}^{\phi} p(1-p)^{|m|-1} \right)^{-1}, \quad \text{si } -\phi \leq h \leq \phi \text{ et } h \neq 0$$

et $v_m^{|h|} = 0$, si $h = 0$, $h < -\phi$ ou $h > \phi$, ceci afin que $\sum_h v_m^{\pm h} = 1$. Nous modélisons un seul événement de mutation à chaque transition du graphe, bien qu'il soit théoriquement possible d'en modéliser plusieurs.

Si $v_m^{\pm h}$ est la probabilité conditionnelle (sachant qu'un événement de mutation se produit) de mutation du marqueur m d'ordre $\pm h$, et si u_m est la probabilité d'une mutation au marqueur m par séquence par génération, la probabilité u d'une mutation par séquence par génération est:

$$\begin{aligned} u &= \sum_{m=1}^L u_m \\ &= \sum_{m=1}^L \sum_{\substack{h=-\phi \\ h \neq 0}}^{\phi} v_m^h u_m. \end{aligned}$$

Notons $M_i^j(l : h)$ une mutation au marqueur m de taille h qui a donné une séquence i à partir d'une séquence j . Pour qu'une séquence i soit le résultat d'une mutation d'une séquence j , ces deux séquences ne doivent différer qu'à un seul marqueur, et à ce marqueur (m), la différence dans le nombre de répétitions des allèles, c'est-à-dire $s_m^i - s_m^j$, doit être inférieure ou égale à ϕ . Une mutation se produit à un locus ancestral m de i ($s_m \in A_i$) avec probabilité $\theta_m = 4Nu_m$, et cette mutation est de taille h avec probabilité v_m^h . Si un tel événement de mutation se produit, alors une séquence i provient d'une séquence j avec probabilité $[v_m^h(n_j + 1)]/n$, car un pas en arrière dans le temps, il y a alors $n_j + 1$ séquences de type j , et le nombre de séquences total reste inchangé. Une mutation de taille h ($h = -\phi, \dots, \phi$, $h \neq 0$) est possible à chaque site ancestral m si $s_m + h$ est entre les valeurs minimum et maximum du locus m , c'est-à-dire si $\delta_h^m = 1$, tel que:

$$\delta_h^m = \begin{cases} 1 & \text{si } \underline{\Delta}_m \leq s_m + h \leq \overline{\Delta}_m, \\ 0 & \text{sinon.} \end{cases}$$

Ainsi, nous obtenons une formule de récurrence similaire à celle que nous avons déduite pour le modèle avec les marqueurs bialléliques à faible taux de

mutation:

$$Q(\mathbf{H}_\tau) = \frac{n(n-1)}{[n(n-1)+n\theta+n\rho]} \left[\sum_{\substack{i, \\ n_i > 1}} \frac{n_i - 1}{n-1} Q(\mathbf{H}_\tau + C_i) \right] \quad \textcircled{1}$$

$$+ 2 \sum_{\substack{i \neq j \\ \text{compatibles}}} \frac{n_k + 1 - \delta_{ik} - \delta_{jk}}{n-1} Q(\mathbf{H}_\tau + C_{ij}^k) \quad \textcircled{2}$$

$$+ \frac{n}{[n(n-1)+n\theta+n\rho]} \left[\sum_i \sum_{\substack{m=1 \\ m \in A_i}}^L \theta_m \sum_{\substack{h=-\phi \\ h \neq 0}}^\phi \delta_h^m v_m^h \frac{n_j + 1}{n} Q(\mathbf{H}_\tau + M_i^j(m : h)) \textcircled{3} \right. \\ \left. + \frac{nL - a}{nL} Q(\mathbf{H}_\tau) \right] \quad \textcircled{4}$$

$$+ \frac{n}{[n(n-1)+n\theta+n\rho]} \left[\sum_i \left\{ \sum_{p=\gamma_i}^{\kappa_i} \rho_p \frac{(n_j + 1)(n_k + 1)}{n(n+1)} Q(\mathbf{H}_\tau + R_i^{jk}(p)) \right\} \textcircled{5} \right. \\ \left. + \frac{nr - b}{nr} Q(\mathbf{H}_\tau) \right], \quad \textcircled{6}$$

où $\underline{\Delta}_m \leq (s \pm \phi)_m \leq \overline{\Delta}_m$ (pour le terme de mutation $\textcircled{3}$). Notons que le facteur n_j dans le terme $\textcircled{3}$ de l'équation ci-dessus dépend de h , qui va alors déterminer le sens de la mutation: expansion ou contraction. Aussi, seules les mutations aux loci ancestraux sont à considérer, donc une mutation $M_i^j(l : h)$ n'est considérée que si le marqueur m de la séquence i est ancestral. Les termes $\textcircled{1} - \textcircled{6}$ réfèrent aux mêmes événements qu'auparavant, soit respectivement, une coalescence de séquences identiques, une coalescence de séquences de types différents, une mutation dans le matériel ancestral, une mutation dans le matériel non ancestral, une recombinaison dans le matériel ancestral, et enfin une recombinaison dans du matériel non ancestral.

L'inférence se fait de la même façon que pour le modèle à mutation non récurrente, mis à part la probabilité de l'ancêtre commun. Dans le modèle du chapitre 3, nous avons supposé que nous connaissions la séquence de l'ancêtre commun: cela était possible puisque les mutations étaient rares. Dans le présent modèle, il est difficile de faire une telle hypothèse. Soit π_m la distribution sta-

tionnaire de P_m la matrice de transition du processus de mutation au marqueur m . La distribution stationnaire peut être trouvée en solutionnant:

$$\boldsymbol{\pi}_m^T = \boldsymbol{\pi}_m^T P_m,$$

où T signifie la transposée d'un vecteur. Notons que dans le cas particulier où $P_m = P$ quel que soit m ($m = 1, \dots, L$), alors $\boldsymbol{\pi}_m = \boldsymbol{\pi}$ quelque soit m . De façon générale, la probabilité que l'ancêtre commun soit (s_1, s_2, \dots, s_L) est:

$$\prod_{m=1}^L \pi_m(s_m),$$

où $\pi_m(s_m)$ est l'entrée s_m du vecteur $\boldsymbol{\pi}_m$. Il nous faut aussi faire une autre distinction par rapport au modèle à mutation non récurrente. Dans ce modèle, le TIM pouvait muter comme les autres marqueurs: les mutations étant non récurrentes, cela était une hypothèse naturelle. Dans le cas de microsatellites, il est difficile de faire une telle hypothèse. Trois solutions sont possibles: (1) autoriser la mutation au TIM, mais à un taux très faible; (2) ne pas autoriser de mutation au TIM et ne considérer que des séquences d'individus atteints; (3) ne pas autoriser de mutation au TIM, tout en considérant des séquences d'individus atteints ou non, mais en stoppant la généalogie quand celle-ci contient deux séquences qui représentent alors les deux ancêtres des deux populations.

Le nombre d'événements de mutation que nous devons considérer dans le cas de microsatellites peut être grand. Ainsi, pour la première étape de l'exemple D, avec $\phi = 3$, plus de 291 événements sont possibles, dont 210 événements de mutation. Plus ϕ est grand, plus le nombre d'événements de mutation à considérer est grand. C'est pourquoi il peut être intéressant de faire une approximation pour les mutations lors de la reconstruction du graphe de recombinaison ancestral, d'une façon similaire à ce que nous avons fait pour les recombinaisons (section 3.11) dans le cas de mutations non récurrentes. Ainsi, nous allons considérer une probabilité de mutation globale par séquence; puis, si un événement de mutation est

choisi pour une certaine séquence, nous choisirons au hasard un des événements de mutation possibles pour cette séquence.

Si une telle mutation $M_i^j(h)$ se produit, nous avons vu que la probabilité de celle-ci est $\theta_m v_m^h$, où θ_m est la probabilité d'une mutation au locus m , et v_m^h est la probabilité que cette mutation soit de taille h . La probabilité de $\mathbf{H}_\tau + M_i^j(h)$ un pas en arrière dans le temps est alors:

$$\sum_{m=1}^L \theta_m \sum_{\substack{h=-\phi \\ h \neq 0}}^{\phi} \frac{n_j + 1}{n} \delta_h^m v_m^h.$$

Ainsi, au temps $\tau + 1$, une transition est faite de \mathbf{H}_τ à $\mathbf{H}_\tau + M_i^j(h)$ avec probabilité:

$$\sum_{m=1}^L \theta_m \sum_{\substack{h=-\phi \\ h \neq 0}}^{\phi} \delta_h^m v_m^h,$$

et on a:

$$f(\mathbf{H}_\tau, \mathbf{H}_{\tau+1}) = \begin{cases} S_{\mathbf{H}_\tau} / D_{\mathbf{H}_\tau} & \text{si } Co \text{ ou } Re \\ (n_j + 1) S_{\mathbf{H}_\tau} / D_{\mathbf{H}_\tau} & \text{si } Mu, \end{cases}$$

où $D_{\mathbf{H}_\tau}$ est défini comme auparavant et $S_{\mathbf{H}_\tau}$ comme suit:

$$\begin{aligned} S_{\mathbf{H}_\tau} &= n \sum_{\substack{i, \\ n_i > 1}} (n_i - 1) + 2n \sum_{\substack{i \neq j \\ \text{compatibles}}} (n_k + 1 - \delta_{ik} - \delta_{jk}) \\ &+ \sum_i \sum_{m=1}^L \theta_m \sum_{\substack{h=-\phi \\ h \neq 0}}^{\phi} \frac{n_j + 1}{n} \delta_h^m v_m^h + \sum_i \sum_{p=\gamma_i}^{\kappa_i} [\rho_p (n_j + 1) (n_k + 1)] / [(n + 1)]. \end{aligned}$$

Le nombre d'événements de mutation à considérer à chaque étape du graphe est alors le nombre de types de séquences différentes. Si nous reprenons la première étape de l'exemple D, le nombre de mutations à considérer passe ainsi de 210 à 15. De plus, cette probabilité ne change pas tant que $\underline{\Delta}_l$ et $\bar{\Delta}_l$ ne changent pas, donc nous n'avons pas besoin de la recalculer à chaque étape.

Nous allons maintenant appliquer ce modèle à deux ensembles de données réelles.

5.2. Analyse des données de l'épilepsie myoclonique progressive (EPM1)

Nous avons déjà présenté l'ensemble D dans le chapitre 3, section 3.7, avec les marqueurs recodés de façon binaire: il s'agit de 88 séquences d'individus atteints d'épilepsie myoclonique progressive. Les données originales sont présentées dans la figure 5.2.1.

0	[65]	5	5	5	5	5
1	[6]	5	5	5	5	7
2	[1]	5	5	5	5	9
3	[2]	5	5	5	5	8
4	[1]	5	5	5	5	6
5	[2]	3	5	5	5	5
6	[2]	4	5	5	5	5
7	[1]	5	5	5	3	9
8	[1]	—	5	5	3	9
9	[1]	5	5	5	4	5
10	[1]	5	5	4	2	7
11	[1]	5	5	2	6	7
12	[2]	5	4	5	5	5
13	[1]	3	4	5	5	5
14	[1]	4	5	6	5	7

Figure 5.2.1. Exemple D de l'épilepsie myoclonique progressive: pour chaque séquence i (0 à 14), multiplicité $[n_i]$, et nombre de répétitions aux cinq marqueurs.

Afin d'analyser ces données, nous devons évaluer κ , le taux de croissance de la population. D'après Virtaneva *et al.* (1996), la population finlandaise a été fondée il y a 2000 ans par 1000 individus, et la population actuelle est d'environ 5 millions. En supposant une croissance exponentielle de la population depuis sa fondation, on obtient la relation

$$1000 = 5 \times 10^6 \exp(-\kappa \times 2000),$$

d'où un taux de croissance annuel donné par:

$$\kappa = -\frac{1}{2000} \ln \left(\frac{1000}{5 \times 10^6} \right) \approx 0.00425.$$

Maintenant, si une génération représente environ 20 ans, le taux de croissance par génération est $20\kappa = 0.085$. En utilisant $2N(0)$ générations comme unité de temps, où $N(0)$ est la taille effective de la population actuelle, évaluée à 10 000, on obtient un taux de croissance par unité de temps $2N(0) \times 0.085 \approx 1700$.

La figure 5.2.2 montre les courbes de vraisemblance obtenues par McPeck et Strahs (1999) et Xiong et Guo (1997). Les deux méthodes placent la mutation dans le bon intervalle, le second, bien que pour la méthode de Xiong et Guo (1997), la valeur du maximum de vraisemblance est presque identique pour $x = 0.4$ dans l'intervalle 2, et $x = 0.53$ dans l'intervalle 3. La méthode de Terwilliger (1995) positionne la mutation dans l'intervalle 3. Ces données ne sont pas faciles à analyser, car la séquence est longue (900 kb) et seulement cinq marqueurs sont disponibles.

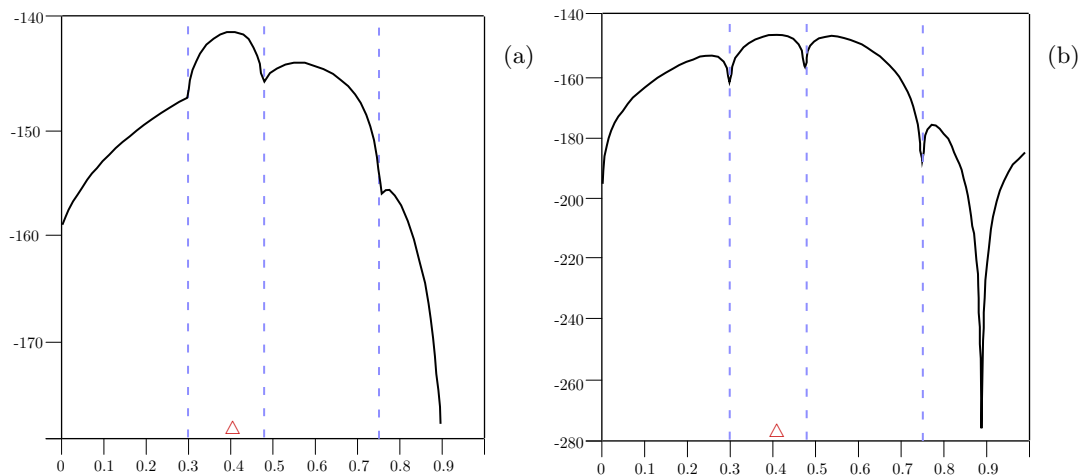


Figure 5.2.2. Courbes de vraisemblance pour l'exemple D: (a) McPeck et Strahs (1999), et (b) Xiong et Guo (1997).

Nous avons essayé d'évaluer la courbe de vraisemblance pour ces données. La figure 5.2.3 présente les résultats de différentes simulations utilisant la méthode originale (a et c) et la méthode approximative (b et d), avec différents taux de mutation, $u_m = 1 \times 10^{-3}$ (a et b) ou $u_m = 1 \times 10^{-4}$ (c et d). Les temps de calculs sont pour (a) de 4j 10h, pour (b) 1j 18h, pour (c) de 7j 6h, et pour (d) de 10j 22h. Rappelons que la mutation cherchée se trouve dans le second intervalle, à la coordonnée 0.33.

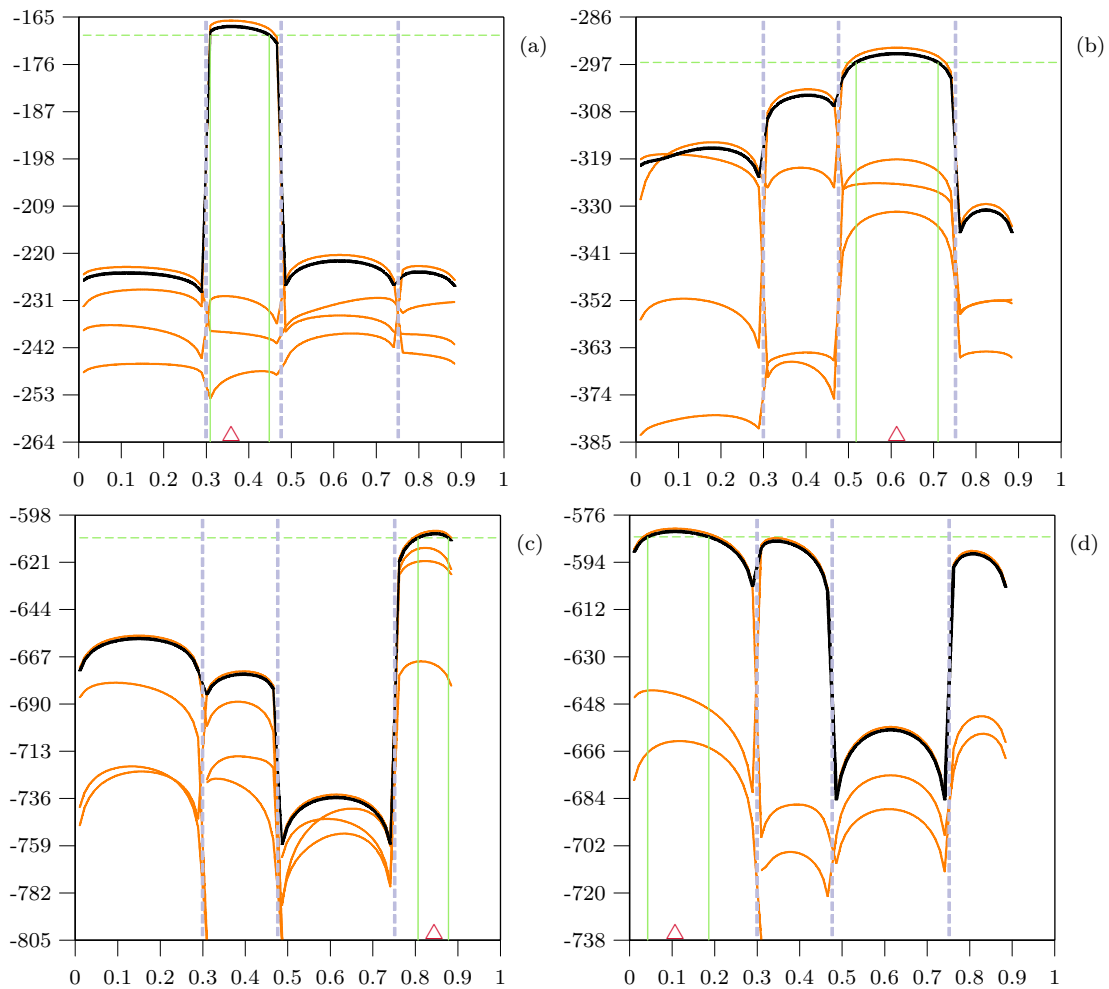


Figure 5.2.3. Courbes de vraisemblance pour l'exemple D: (a) méthode originale avec 500m d'itérations chaque et $u_m = 1 \times 10^{-3}$, (b) méthode approximative avec 1M d'itérations chaque et $u_m = 1 \times 10^{-3}$, (c) méthode originale avec 500m d'itérations chaque et $u_m = 1 \times 10^{-4}$, (d) méthode approximative avec 1M d'itérations chaque et $u_m = 1 \times 10^{-4}$.

Comme on peut le voir dans la figure 5.2.3, les courbes de vraisemblance montrent beaucoup de variations. Il est très difficile de tenter de les interpréter.

5.3. Analyse des données de la fibrose kystique

L'exemple E est celui de la fibrose kystique. Les données ont été publiées dans Kerem *et al.*, (1989). Elles sont constituées de 94 séquences d'individus atteints, et 92 séquences d'individus non atteints, avec 23 marqueurs bialléliques couvrant une région de 1.8Mb (cf. fig. 5.3.1). Les distances physiques sont converties en

distances génétiques tel que $0.5cM = 1Mb$ et nous utiliserons un taux de mutation $u_l = 2.5 \times 10^{-5}$ (comme dans Morris *et al.*, 2002). La mutation $\Delta F508$ causant la maladie est maintenant connue, se situant à une distance de $0.88Mb$ du début de la séquence. La mutation est responsable d'environ 66% des personnes souffrant de la maladie. Parmi les 94 séquences provenant d'individus atteints, on sait que seulement 62 d'entre elles possèdent la mutation $\Delta F508$.

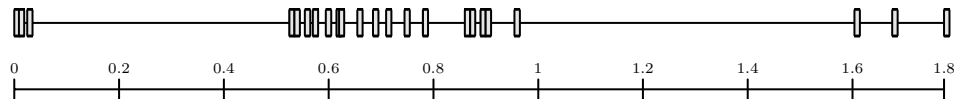


Figure 5.3.1. Répartition des 23 marqueurs, représentés par des barres verticales grises, sur une échelle en Mb.

Cet ensemble de données est intéressant à plus d'un titre: il a été très étudié, et nous pouvons ainsi comparer les résultats de notre méthode pour cet exemple avec la plupart des autres méthodes de cartographie fine. De plus, la mutation ne réside pas dans la région centrale où le déséquilibre de liaison est le plus élevé (voir Morris *et al.*, 2002), et enfin, le marqueur le plus proche de la mutation montre une faible association avec $\Delta F508$. Les données sont présentées dans l'annexe A.

La mutation de la fibrose kystique est apparue il y a environ 200 générations, soit voilà environ 4000 ans, et le nombre de chromosomes porteurs est maintenant de 2×10^7 (Xiong et Guo, 1997). En supposant une croissance exponentielle des chromosomes porteurs depuis leur apparition, on obtient la relation:

$$1 = 2 \times 10^7 \exp(-\kappa \times 4000)$$

d'où un taux de croissance annuel donné par:

$$\kappa = -\frac{1}{4000} \ln \left(\frac{1}{2 \times 10^7} \right) \approx 0.00420.$$

Toujours en prenant une génération de 20 ans, le taux de croissance par génération est 20κ . Si $N(0)$ est la taille effective de la population actuelle, évaluée à 10 000, le taux de croissance par unité de temps est $2N(0) \times 20 \times \kappa \approx 1681$.

Le tableau 5.3.1 nous montre les estimations obtenues par différentes méthodes sur ces données. Les estimés de r_T varient beaucoup entre les différentes méthodes, et les intervalles de confiance aussi: les intervalles de confiance incluent entre 6% (Liu *et al.*, 2001) et 56% (McPeck et Strahs, 1999) de la séquence. Un intervalle couvrant plus de la moitié de la séquence est à toute fin pratique d’une utilité limitée. Certaines des méthodes se sont contentées, comme nous essaierons de le faire également, de limiter l’échantillon aux cas “réels” afin d’éliminer les problèmes de pénétrance: c’est-à-dire que l’on ne tient pas compte dans l’analyse des individus atteints qui ne sont pas porteurs de $\Delta F508$.

Tableau 5.3.1. Estimés de r_T pour les données de la fibrose kystique par différentes méthodes de cartographie fine (IC: intervalle de confiance, %IC: proportion de l’IC par rapport à la séquence complète. Adapté de Morris *et al.*, 2002) .

Méthode	Estimé	IC	%IC	Note
Terwilliger (1995)	.77	.69–.97	15%	IC à 99%; ne contient pas $\Delta F508$
Xiong et Guo (1997)	.80	-	-	Porteur de $\Delta F508$ seulement
Collins et Morton (1998)	.83	-	-	Porteur de $\Delta F508$ seulement
McPeck et Strahs (1999)	.95	.44–1.46	56%	
Morris <i>et al.</i> (2000)	.80	.61–1.07	25%	
Liu <i>et al.</i> (2001)	-	.82–.93	6%	
Morris <i>et al.</i> (2002)	.86	.65–1.04	22%	Porteur de $\Delta F508$ seulement
Morris <i>et al.</i> (2002)	.85	.65–1.00	19%	

Nous reportons dans la figure 5.3.2 les courbes de vraisemblance de McPeck et Strahs (1999) (a) et Xiong et Guo (1997) (b), mis sur une échelle semblable à la nôtre de façon à ce que l’on puisse faire des comparaisons. Notons que nous montrons ces courbes de vraisemblance uniquement parce que ce sont les seules disponibles; les autres méthodes ne les montrent pas, ou utilisent des distributions *a posteriori* qui sont difficiles à comparer avec des courbes de vraisemblance.

Nous allons maintenant présenter les analyses que nous avons effectuées sur ces données. Nous devons préciser que ces analyses sont basées sur un faible nombre d’itérations, et il faut prendre des précautions quant à l’interprétation

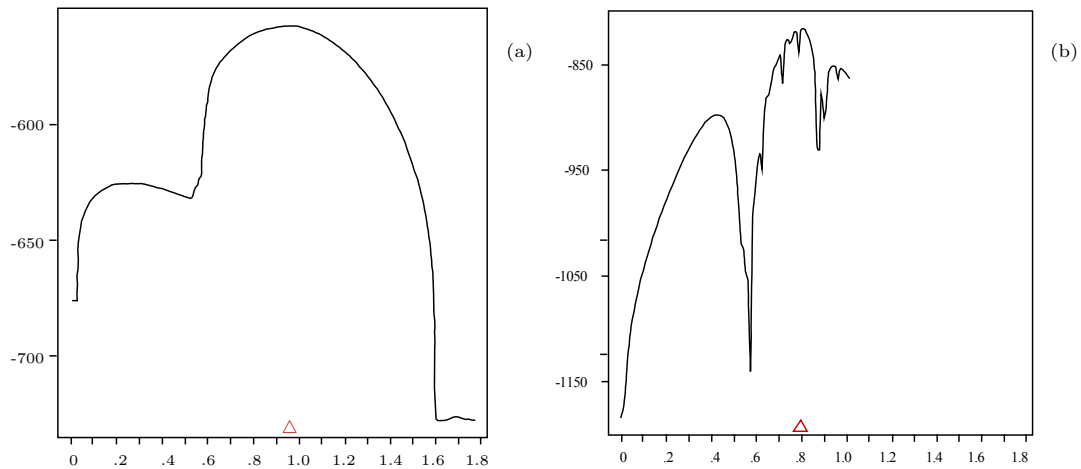


Figure 5.3.2. Courbes de vraisemblance pour l'exemple E: (a) McPeck et Strahs (1999), et (b) Xiong et Guo (1997) (porteurs de $\Delta F508$ seulement).

des résultats. Ceux-ci ne peuvent être considérés que comme préliminaires. De plus, nous avons choisi de n'inclure dans l'analyse que des cas, ceci pour deux raisons: d'abord diminuer le temps de calcul, et ensuite régler de façon simple la question de la mutation au TIM que nous avons discutée dans la section précédente. Aussi, nous avons utilisé la méthode approximative pour les recombinaisons et mutations tel que discutées auparavant. En effet, nous ne disposons pas d'outils de calcul assez puissants pour appliquer la méthode originale.

Commençons par calculer la courbe de vraisemblance uniquement chez les cas porteurs de la mutation $\Delta F508$. Deux simulations de 100m d'itérations ont été effectuées. Chacune d'entre elles nécessite environ deux jours de calcul. Les estimés sont de 0.92Mb dans le premier cas, et de 0.865Mb dans le second cas. Les estimés sont plutôt bons, puisque très proches de la vraie position de r_T . Si l'on compare avec les courbes d'autres méthodes (cf. fig. 5.3.2), le pic de vraisemblance est plus clair. Notons cependant que ces courbes sont assez variables, puisque même si l'on obtient des pics de vraisemblance très proches de la mutation, les deux estimés ne sont pas dans le même intervalle. L'allure générale des deux courbes de vraisemblance est quand même très similaire: une région (centrale) montre un certain déséquilibre de liaison avec $\Delta F508$. Notons que la mutation cherchée

n'est pas au centre de la région montrant le plus de déséquilibre de liaison.

Comme les simulations sont longues à effectuer, et que seulement une région semble "plus intéressante", une stratégie consiste à n'utiliser que la région centrale de la séquence pour l'analyse. Nous ignorons ainsi une partie de l'information,

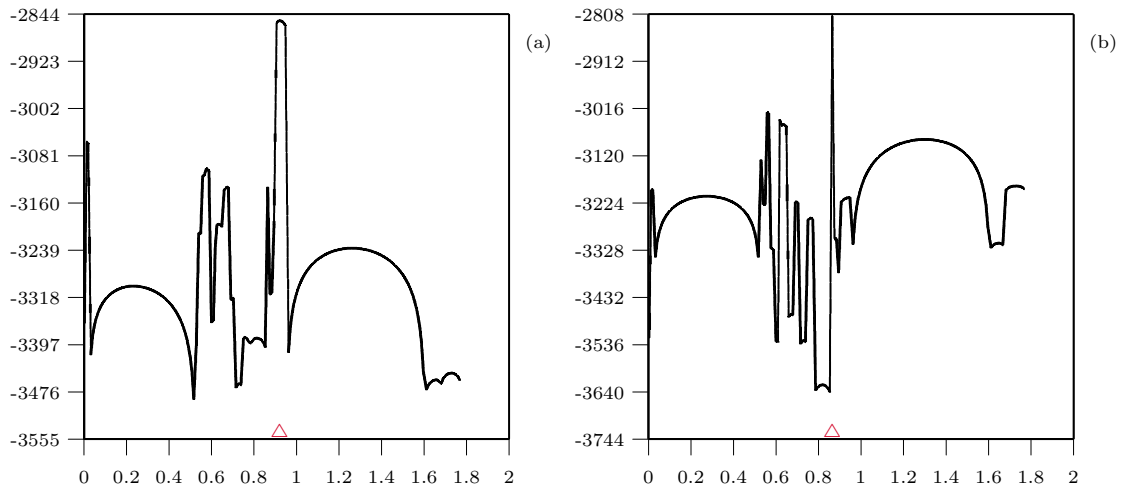


Figure 5.3.3. Deux courbes de vraisemblance pour l'exemple E, basées sur 100m d'itérations, en incluant seulement les séquences porteuses de $\Delta F508$.

mais nous pouvons faire plus de simulations, dans l'espoir d'avoir moins de variation dans les courbes de vraisemblance. Nous avons donc essayé cette stratégie en incluant cette fois-ci tous les cas. Les résultats sont présentés dans la figure 5.3.4. Dans la partie (a) de la figure, deux courbes de 100m d'itérations, et dans la partie (b), deux courbes de 250m d'itérations. Environ 9 jours de calcul sont nécessaires pour obtenir les quatre courbes de la figure 5.3.4. Évidemment, on voit beaucoup de variations dans les courbes de vraisemblance, mais un profil similaire se dégage: un pic central (proche de la coordonnée 0.74) fort, et un second pic proche de $\Delta F508$. Notons que la forme de la vraisemblance est similaire à celle de Xiong and Guo (1997) (cf. fig. 5.3.2 (b)), la vraisemblance augmente aux mêmes positions de façon semblable, bien que les cas non porteurs de $\Delta F508$ soient exclus de leurs analyses.

Il serait évidemment intéressant de faire plus de simulations, afin de voir si le pic de vraisemblance proche de $\Delta F508$ se montre plus clairement ou pas. Mais

il est probable, compte tenu de ce que nous avons déjà vu dans cette thèse sur la variabilité des courbes de vraisemblance produites par la méthode, que nous aurons toujours une variabilité assez marquée. Il est surprenant de voir que l'on obtienne quand même des courbes de vraisemblance ayant la même allure générale,

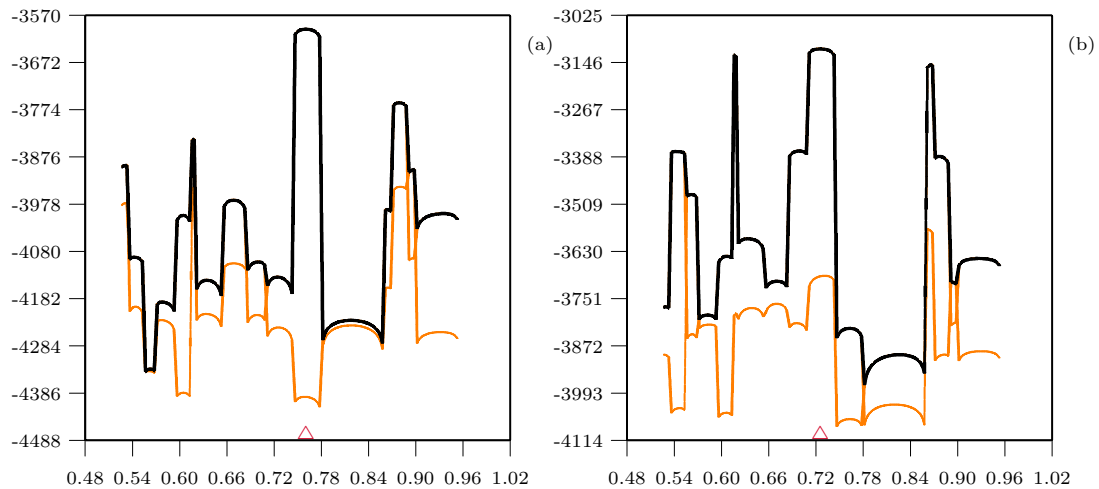


Figure 5.3.4. Courbes de vraisemblance pour l'exemple E sur une partie seulement de la séquence, basées sur (a) 100m d'itérations chaque, (b) 250m d'itérations chaque.

même si un faible nombre d'itérations est utilisé. Nous avons déjà aperçu cela dans la section 3.9, dans laquelle nous avons vu qu'avec un faible nombre d'itérations, nous obtenions non seulement des courbes de vraisemblance similaires, mais que l'allure générale de ces courbes était assez proche des courbes obtenues avec plusieurs dizaines de millions d'itérations.

5.4. Nombre maximum d'itérations dans un graphe

Dans l'implantation de la méthode, il y a une option pour arrêter la construction de graphes trop longs (selon le nombre d'étapes). Quand la construction d'un graphe est stoppée de cette manière, la vraisemblance associée à ce graphe devient nulle, autrement dit le poids de ce graphe est 0, mais on tient évidemment compte de "l'annulation" de ce graphe dans le calcul de la vraisemblance. Soit ν le nombre maximum d'événements permis pour la construction d'un graphe.

Dans l'application de la méthode, nous prenons une valeur tellement grande de ν qu'en pratique la construction d'un graphe n'est jamais arrêtée. Notons que le résultat du programme précise le nombre de graphes ainsi abandonnés.

Dans la construction des graphes, il y a de grandes différences entre les poids des différents graphes, de telle sorte que seulement une faible proportion des graphes contribuent réellement à la vraisemblance. Bien qu'il n'existe pas de relation directe entre le poids d'un graphe et le nombre d'étapes nécessaires à sa construction, nous pouvons tenter d'améliorer la vitesse du programme en arrêtant tôt la construction des graphes contenant trop d'étapes. Notons qu'un procédé similaire est utilisé dans Griffiths et Marjoram (1996), mais le procédé pour arrêter la construction des graphes n'est pas clairement précisé.

Une des façons les plus simples d'évaluer l'effet d'une telle méthode est de la simuler. Nous prendrons trois exemples, et pour chacun d'eux, nous calculerons la vraisemblance de façon usuelle, mais aussi pour différents niveaux de ν , et ceci sur les mêmes graphes. Soit $\bar{\nu}$ le pourcentage des graphes dont le nombre d'étapes est inférieur ou égal à ν . Nous allons choisir 10 valeurs de ν correspondant approximativement à des valeurs de $\bar{\nu}$ de 90%, 80%, 70%, 60%, 50%, 40%, 30%, 20%, 10%, et 5%. Le tableau 5.4.2 présente pour chacun des trois exemples, les valeurs de ν , les valeurs correspondantes de $\bar{\nu}$, et les temps de calcul qui en découlent. La dernière ligne du tableau est une référence afin de pouvoir comparer les temps de calcul; cette référence est calculée avec une grande valeur de ν de telle sorte que $\bar{\nu} = 100\%$. Notons que pour l'exemple D, la référence utilisée est $\bar{\nu} \approx 70\%$; pour cet exemple, ρ est très grand, les graphes peuvent être très longs, et la version rapide du programme est utilisée (cf. section 4.7), ce qui a pour conséquence que 30% des graphes ont des poids inférieurs à la précision de la machine, et ne peuvent être calculés. Dans la colonne du temps, on trouvera un petit graphique indiquant le gain de temps que l'on réalise à utiliser une valeur particulière de ν : une barre complètement noire indique la référence, et une barre entièrement blanche exprime que deux fois moins de temps est nécessaire pour

faire le même nombre d'itérations. Les estimations de $\bar{\nu}$ sont basées sur 5M de simulations pour les exemples B et C, et sur 500m simulations pour l'exemple D; notons que ces proportions sont très stables, quel que soit le nombre de simulations. Ensuite, 10 évaluations de la vraisemblance ont été effectuées pour chacune des valeurs de ν , et une autre évaluation pour servir de référence de telle sorte que l'on ait $\bar{\nu} = 100\%$.

Tableau 5.4.2. Évaluation du pourcentage de graphes rejetés ($\bar{\nu}$) et du gain de temps de calcul en modifiant le nombre d'itérations permis (ν) pour reconstruire un graphe pour les exemples B, C et D.

B			C			D		
ν	$\bar{\nu}$	Temps	ν	$\bar{\nu}$	Temps	ν	$\bar{\nu}$	Temps
45	91.2	7m32s	136	91.1	9m8s	.	.	.
40	77.9	7m20s	131	83.0	9m5s	.	.	.
37	71.0	7m10	128	70.1	8m47s	.	.	.
35	62.7	6m59s	125	61.9	8m46s	630	61.34	15m55s
33	53.2	6m47s	123	52.6	8m27s	515	49.31	14m22s
31	42.7	6m32s	121	42.7	8m18s	442	40.07	13m6s
30	31.9	6m25s	120	32.8	8m11s	378	30.41	11m46s
27	21.7	5m54s	118	23.6	7m56s	319	19.92	10m34s
25	13.0	5m31s	114	9.5	7m24s	263	9.41	9m4s
23	6.4	5m32s	112	5.1	7m7s	232	4.39	8m4s
Ref	100	7m46s	Ref	100	9m9s	9000	69.07	16m59s

Le nombre moyen d'étapes nécessaires pour la construction des graphes est 33.5 ± 8.1 pour l'exemple B, 123.3 ± 8.4 pour C, et $424.5 \pm 143,6$ pour D (moyennes calculées sur des graphes construits sans contrainte). On peut voir que le gain de temps commence à être appréciable seulement pour de petites valeurs de ν , c'est-à-dire quand on stoppe très tôt la construction des graphes. Le gain est de l'ordre d'une fois et demi pour les deux premiers exemples, mais le gain peut être fort appréciable (deux fois) pour le dernier exemple.

La figure 5.4.1 montre les évaluations de la même vraisemblance pour l'exemple B effectuées pour les différentes valeurs de ν du tableau 5.4.2. Nous avons pour cela modifié le programme afin qu'il calcule la vraisemblance selon ν ; ainsi,

on peut apprécier exactement l'effet de cette approximation. Onze courbes sont représentées sur chacun des graphiques (a) et (b), mais certaines ne sont pas visibles, car elles sont surimposées sur la courbe la plus foncée, qui est la courbe référence. Chacune de ces courbes est une évaluation de la même vraisemblance. Les trois courbes les plus basses de (a) correspondent respectivement aux valeurs de $\bar{\nu}$ de 5%, 10% et 20%. Plus $\bar{\nu}$ est petit, plus le nombre de graphes abandonnés est grand, et plus la vraisemblance est basse. Il est intéressant de noter que les courbes correspondant aux valeurs de $\bar{\nu}$ supérieures à 30% sont assez semblables à la courbe de référence. À $\bar{\nu} = 30\%$, cela signifie qu'environ 70% des graphes sont abandonnés, donc que seulement 30% d'entre eux contribuent réellement à la vraisemblance. Si l'on observe la figure (b) qui correspond à 5M d'itérations, on observe qu'à partir de $\bar{\nu} \approx 70\%$, la vraisemblance est la même. Elle montre donc un profil légèrement différent. Seules les courbes correspondant aux valeurs 70%, 80% et 90% de $\bar{\nu}$ sont similaires à la courbe de référence. Cela signifie que 70% des graphes contribuent à la vraisemblance. Il est intéressant de noter que les courbes pour des valeurs de $\bar{\nu}$ d'au moins 20% sont quand même assez proches de la courbe de référence, en ce qui concerne la hauteur et la position du maximum.

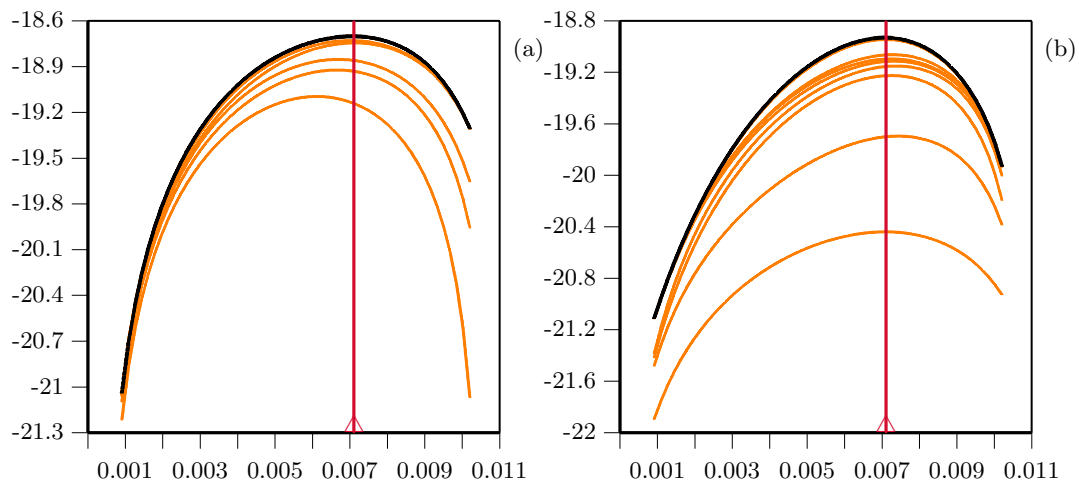


Figure 5.4.1. Courbes de vraisemblance pour l'exemple B, pour les 10 niveaux de ν du tableau 5.4.2: a) 1M d'itérations, b) 5M d'itérations.

Les figures 5.4.2 (a) et (b) présentent des profils très similaires pour l'exemple C: seules les courbes correspondant aux valeurs de 5% et 10% de $\bar{\nu}$ sont visibles: les 8 autres courbes sont confondues avec la courbe de référence. On peut donc en déduire que l'on pourrait utiliser une valeur de $\bar{\nu}$ de 20% sans affecter l'estimation, c'est-à-dire omettre 80% des graphes.

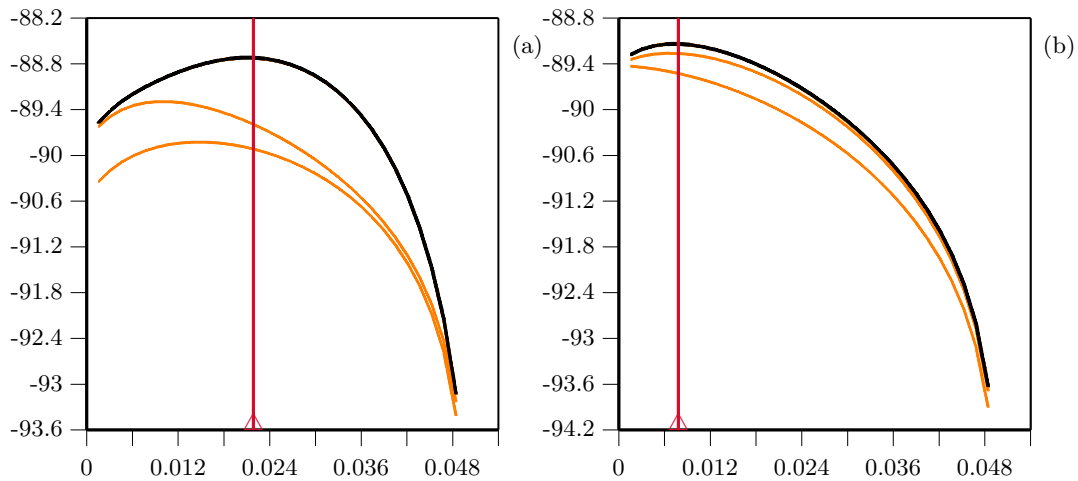


Figure 5.4.2. Courbes de vraisemblance pour l'exemple C, pour les 10 niveaux de ν du tableau 5.4.2: a) 1M d'itérations, b) 5M d'itérations.

Pour notre dernier exemple (D), nous avons un profil un peu différent: pour les deux cas, 100m et 1M d'itérations, toutes les sept courbes sont confondues avec notre courbe de référence; la vraisemblance est ainsi la même pour $\bar{\nu} \approx 70\%$ que pour $\bar{\nu} \approx 5\%$, ce qui veut dire que peu de graphes, moins de 5%, contribuent en fait à la vraisemblance (cf. fig. 5.4.3 (a) et (b)). Ceci peut s'expliquer en partie par le type de l'exemple: ce sont des données réelles, et la distribution proposée qui construit les graphes est probablement moins efficace dans ce cas: ainsi, on génère beaucoup de graphes ayant un grand nombre d'étapes et un petit poids associé, et peu de graphes courts ayant des poids plus grands. La vraisemblance étant définie par la moyenne des poids, on voit ainsi que si l'on abandonne en cours d'évaluation les graphes ayant un grand nombre d'étapes, et qu'on leur assigne un poids nul, cela ne fait pratiquement aucune différence, c'est comme si on avait continué la construction des ces graphes jusqu'au MRCA afin d'avoir une juste évaluation de la vraisemblance. Donc, à toute fin pratique, le poids de

ces longs graphes abandonnés est nul par rapport au poids des autres graphes.

En conclusion, on constate que peu de graphes contribuent en fait à la vraisemblance. L'utilisation de cette option peut nous faire gagner un temps important. On a vu que pour l'ensemble de données D, par exemple, nos estimations peuvent être obtenues deux fois plus rapidement en utilisant une valeur de ν

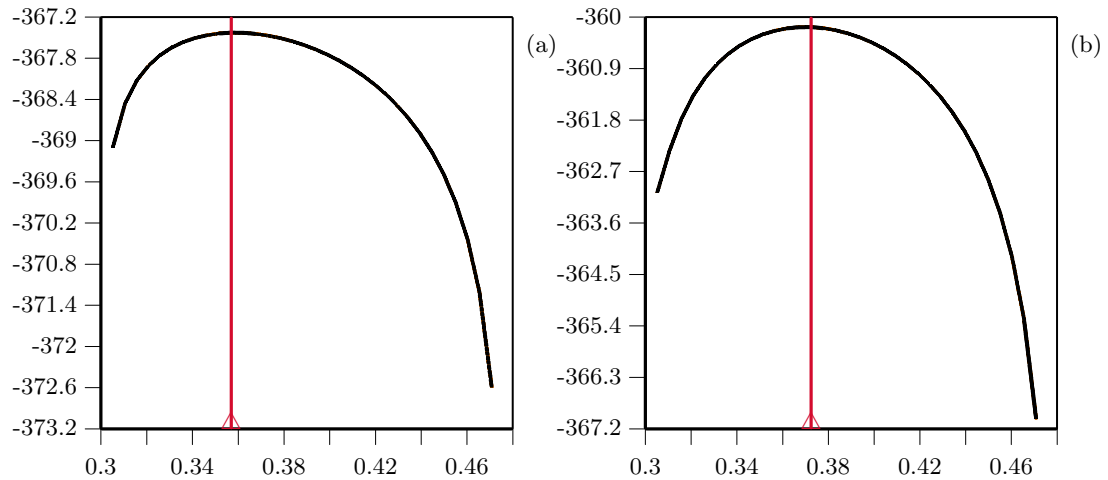


Figure 5.4.3. Courbes de vraisemblance pour l'exemple D, pour les 7 niveaux de ν du tableau 5.4.2: a) 100m d'itérations, b) 1M d'itérations..

menant à $\bar{\nu} \approx 5\%$. Compte tenu que plusieurs jours, voire plusieurs semaines, peuvent être nécessaires pour obtenir des résultats stables, il est plus que tentant de restreindre le nombre d'étapes des graphes. La valeur de $\bar{\nu}$ à utiliser n'est cependant pas facile à déterminer. Plusieurs essais comme l'on vient de le faire montrent assez clairement la valeur maximum à ne pas utiliser.

5.5. Ajustement des poids des recombinaisons

Un autre paramètre que nous pouvons utiliser pour améliorer la vitesse d'exécution consiste à modifier les probabilités des recombinaisons. Nous savons que plus les séquences sont grandes, plus long sera le temps nécessaire pour relier les séquences observées à l'ancêtre commun. On introduit alors le facteur ξ qui est un

ponds de recombinaison. Commençons par diviser puis multiplier par ξ la probabilité de recombinaison dans l'équation (3.1), de telle sorte que la probabilité de l'événement $R_i^{jk}(p)$ est alors:

$$\xi \rho_p \frac{(n_j + 1)(n_k + 1)}{(n + 1)S_{\mathbf{H}_\tau}},$$

et la fonction $f(\mathbf{H}_\tau, \mathbf{H}_{\tau+1})$ associée à cet événement est maintenant:

$$\frac{S_{\mathbf{H}_\tau}}{\xi D_{\mathbf{H}_\tau}}.$$

Nous avons essayé différentes valeurs de ξ sur trois exemples. Les figures 5.5.1,

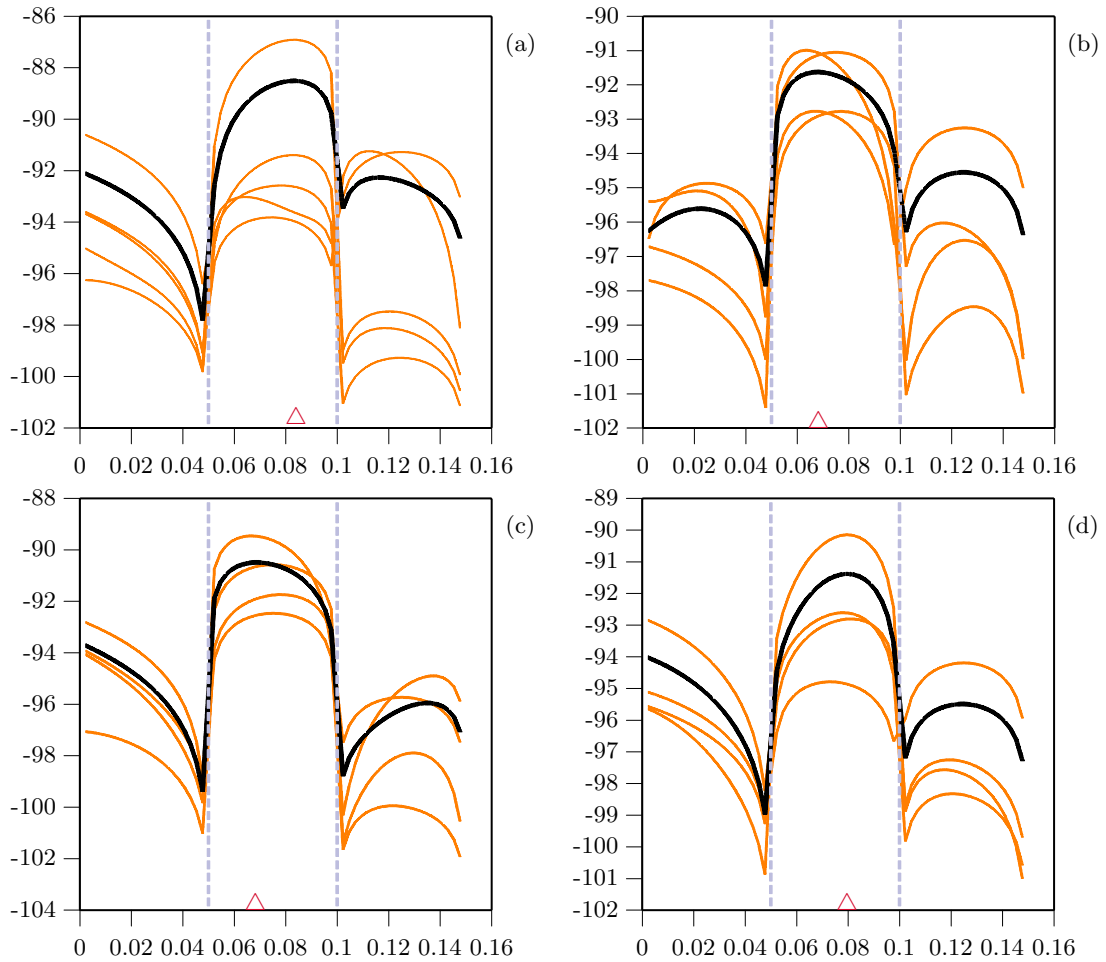


Figure 5.5.1. Courbes de vraisemblance pour l'exemple A avec 1M d'itérations et leur courbe cumulée, pour (a) $\xi = 1$, (b) $\xi = 0.85$, (c) $\xi = 0.70$ et (d) $\xi = 0.55$.

5.5.2 et 5.5.3 qui suivent montrent l'estimation originale, c'est-à-dire quand $\xi = 1$ (a), et les estimations faites avec $\xi = 0.85$ (b), $\xi = 0.70$ (c) et $\xi = 0.55$. Pour pouvoir comparer les estimations, notons que le nombre d'itérations est le même pour chacune des valeurs de ξ pour un même exemple.

Les courbes de vraisemblance avec les différentes valeurs de ξ ainsi que la courbe de référence ($\xi = 1$) sont présentées pour l'exemple A dans la figure 5.5.1. On peut voir que les estimations ne semblent pas varier en fonction de ξ . La forme générale des courbes de vraisemblance est stable pour les valeurs de ξ testées, et la variation entre les différents profils ne semble pas augmenter au fur

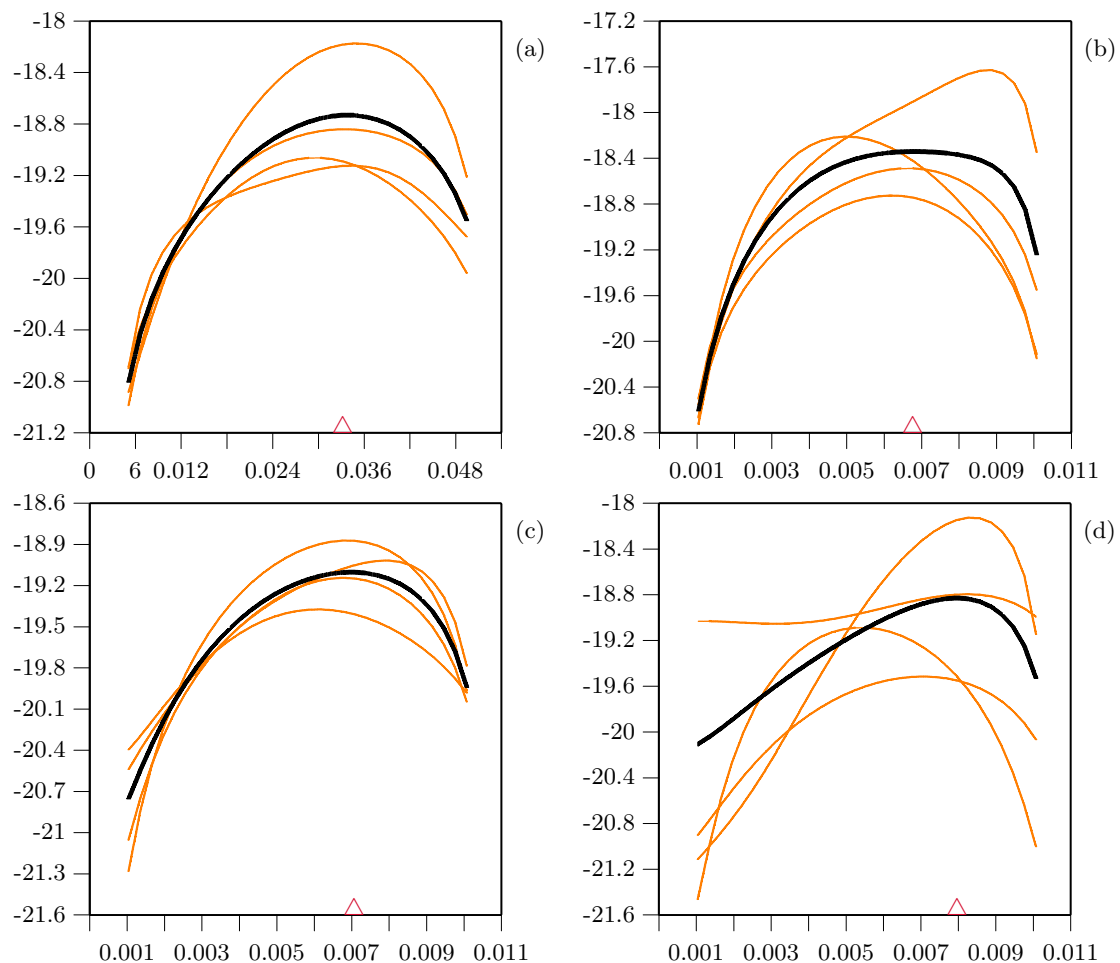


Figure 5.5.2. Courbes de vraisemblance pour l'exemple B avec 5M d'itérations et leur courbe cumulée, pour (a) $\xi = 1$, (b) $\xi = 0.85$, (c), $\xi = 0.70$ et (d) $\xi = 0.55$.

et à mesure que ξ diminue. Notons que les gains dans le temps d'exécution sont respectivement de l'ordre de 6%, 15% et 21% pour $\xi = 0.85$, $\xi = 0.70$ et $\xi = 0.55$. Si de nombreuses itérations sont nécessaires pour obtenir la convergence, un gain de 20% n'est pas négligeable.

La figure 5.5.2 présente les courbes de vraisemblances sur 5M d'itérations pour le second exemple. Bien qu'une des quatre estimations pour $\xi = 0.85$ soit assez mauvaise, les courbes s'approchent de très près des courbes de référence, et l'estimation elle-même varie peu. Par contre, pour $\xi = 0.55$, la variation des courbes augmente considérablement. Le gain dans les temps d'exécution pour les

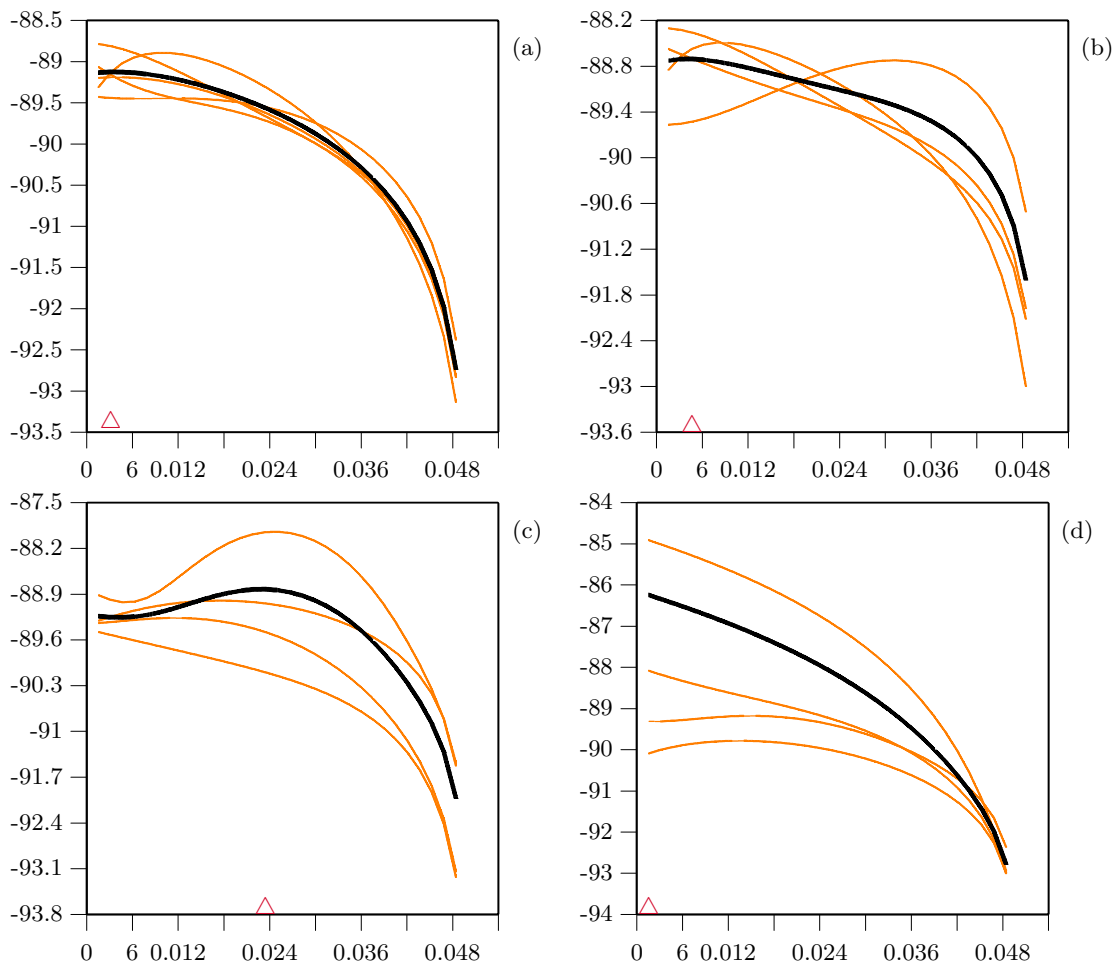


Figure 5.5.3. Courbes de vraisemblance pour l'exemple C avec 5M d'itérations et leur courbe cumulée, pour (a) $\xi = 1$, (b) $\xi = 0.85$, (c) $\xi = 0.70$ et (d) $\xi = 0.55$.

trois valeurs de ξ sont minimales pour cet exemple: 3%, 6% et 9%. Compte tenu de la variation créée par l'utilisation d'un poids de recombinaison différent dans ce cas et du peu de temps gagné, il semble que la variation proposée n'apporte rien pour cet ensemble de données. Le peu de gain dans le temps d'exécution s'explique peut-être par la petite valeur de ρ , soit ici 5.

Les courbes de vraisemblance pour l'exemple C sont présentées dans la figure 5.5.3. On voit clairement que plus ξ diminue, plus la variation des courbes est grande. Ici encore, le gain dans le temps d'exécution est assez faible: entre moins de 1% et 9%. L'estimation devient même catastrophique à partir de $\xi = 0.70$.

Comme on vient de le voir, l'utilisation d'un poids de recombinaison différent pourrait être une option, mais une vérification du gain de temps d'exécution ainsi obtenu est recommandée (ceci est très facile à réaliser, en quelques minutes seulement, en utilisant une option du programme pour estimer des temps de calcul pour faire un certain nombre d'itérations). Pour l'exemple D (non montré), on a estimé que le gain de temps pour $\xi = 0.55$ est de 20%. Comme nous l'avons vu, les simulations sur ces données (exemple D) sont coûteuses en temps d'exécution, et l'utilisation d'un ξ pourrait être une solution avantageuse. Malheureusement, le choix d'une valeur pour ξ est difficile. S'il est trop petit, peu de recombinaisons se produiront, et la courbe de vraisemblance de r_T sera peu courbée. Aussi, une grande prudence serait nécessaire quant à l'interprétation des résultats. Notons cependant à ce propos que si les profils de différentes estimations sont très similaires, c'est un gage de convergence des estimations, et ce, pour n'importe quelle valeur de ξ . L'utilisation de ce paramètre dans les exemples que nous avons montrés est bien loin de confirmer que cette modification est avantageuse, mais cette possibilité existe.

5.6. Recombinaisons et sites ancestraux

Bien que cela ne soit pas indiqué dans Griffiths et Marjoram (1996), nous savons, grâce à une communication personnelle de R.C. Griffiths, que dans

l'implantation originale qu'ils ont faite, les sites qui perdent leur état polymorphe sont retirés des séquences ancestrales, et que la reconstruction se poursuit sans ces sites. La figure 5.6.1 illustre ce concept par un exemple: sur la partie gauche, le graphe tel que reconstruit par notre méthode usuelle, et sur la partie droite le même graphe correspondant reconstruit avec la modification proposée.

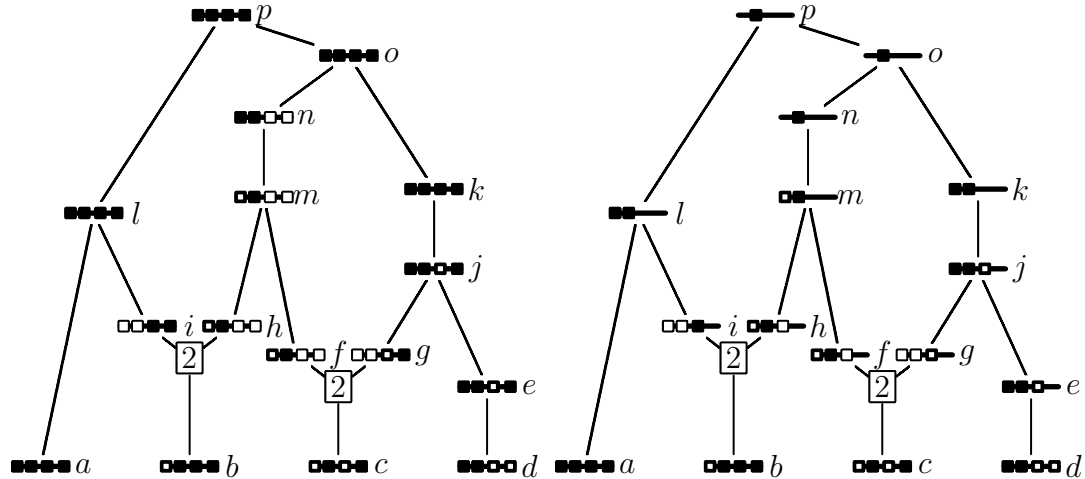


Figure 5.6.1. Exemple de graphe de recombinaison ancestral: à gauche avec la méthode usuelle, et à droite en supprimant les loci qui perdent leur état polymorphe.

La modification consiste à exclure des séquences les loci ayant mutés: par exemple, à la première étape du graphe, on a une mutation de la séquence d vers la séquence e : le dernier site est alors ignoré pour toutes les séquences ancestrales, et la construction du graphe continue. Nous avons essayé d'inclure cette modification dans notre programme. Il est malheureusement difficile de le faire directement compte tenu des contraintes techniques, mais il n'est pas difficile de simuler un processus équivalent. Il suffit de restreindre certains événements de recombinaison, en fonction des événements de mutation ayant eu lieu. À chaque événement de mutation, nous mettons à jour un vecteur nous informant des sites où les mutations ont eu lieu (M), et nous interdisons certaines recombinaisons selon la règle qui suit. Soit M_m le $m^{\text{ème}}$ élément de M ($m = 1 \dots, L$) tel que $M_m = 1$ si une mutation a eu lieu dans une séquence au locus m , et $M_m = 0$ sinon, et R_p le $p^{\text{ème}}$ élément de R ($p = 1 \dots, L - 1$) tel que $R_p = 1$ si une recom-

binisation est autorisée dans l'intervalle p , et 0 sinon. En partant du début de la séquence, si $M_1 = 0$, donc $R_p = 1$ pour tout p ; si $M_1 = 1$, alors $R_1 = 0$; ensuite, si $M_1 = 1$ et $M_2 = 1$, alors $R_2 = 0$, mais si $M_2 = 0$ alors $R_2 = 1$, et ainsi de suite. Nous faisons la même chose en partant de la fin de la séquence. La figure 5.6.2 illustre la procédure par un exemple.

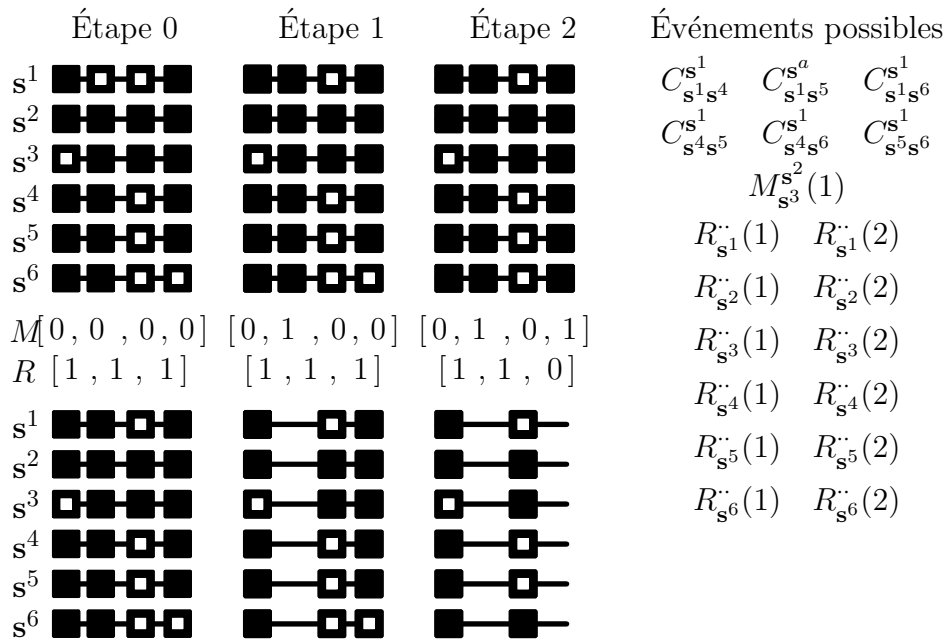


Figure 5.6.2. Exemple fictif illustrant la procédure qui interdit certaines recombinaisons. Les événements possibles à l'étape 2 sont à droite. M est le vecteur informatif des mutations, et R indique les intervalles où les recombinaisons sont permises.

Dans l'exemple de la figure 5.6.2, nous avons 6 séquences. Le premier événement est une mutation au marqueur 2 de la séquence s^1 , puis le second événement un mutation du marqueur 4 de la séquence s^6 . Comme on peut le voir, le fait de supprimer les sites mutés ne change rien aux autres événements: les mêmes coalescences et les mêmes mutations sont possibles. Ce changement n'influe que sur les événements de recombinaison. La figure 5.6.3 qui suit montre l'effet de la procédure qui interdit certaines recombinaisons sur les estimations. Les parties (a) et (b) comparent des courbes de vraisemblance basées sur 1M d'itérations avec la méthode originale et avec la procédure décrite ci-dessus, et les parties (c)

et (d) comparent de la même façon les courbes de vraisemblance basées sur 5M d'itérations. Bien que les estimés obtenus par les deux méthodes soient similaires, on peut voir que la modification entraîne une grande variabilité des courbes de vraisemblance, et cela ne semble pas changer beaucoup si le nombre d'itérations augmente.

La figure 5.6.4 illustre la même chose, mais avec les données de l'exemple C. Les mêmes remarques s'appliquent ici, bien que la variation créée par la procédure limitant les recombinaisons soit moins grande.

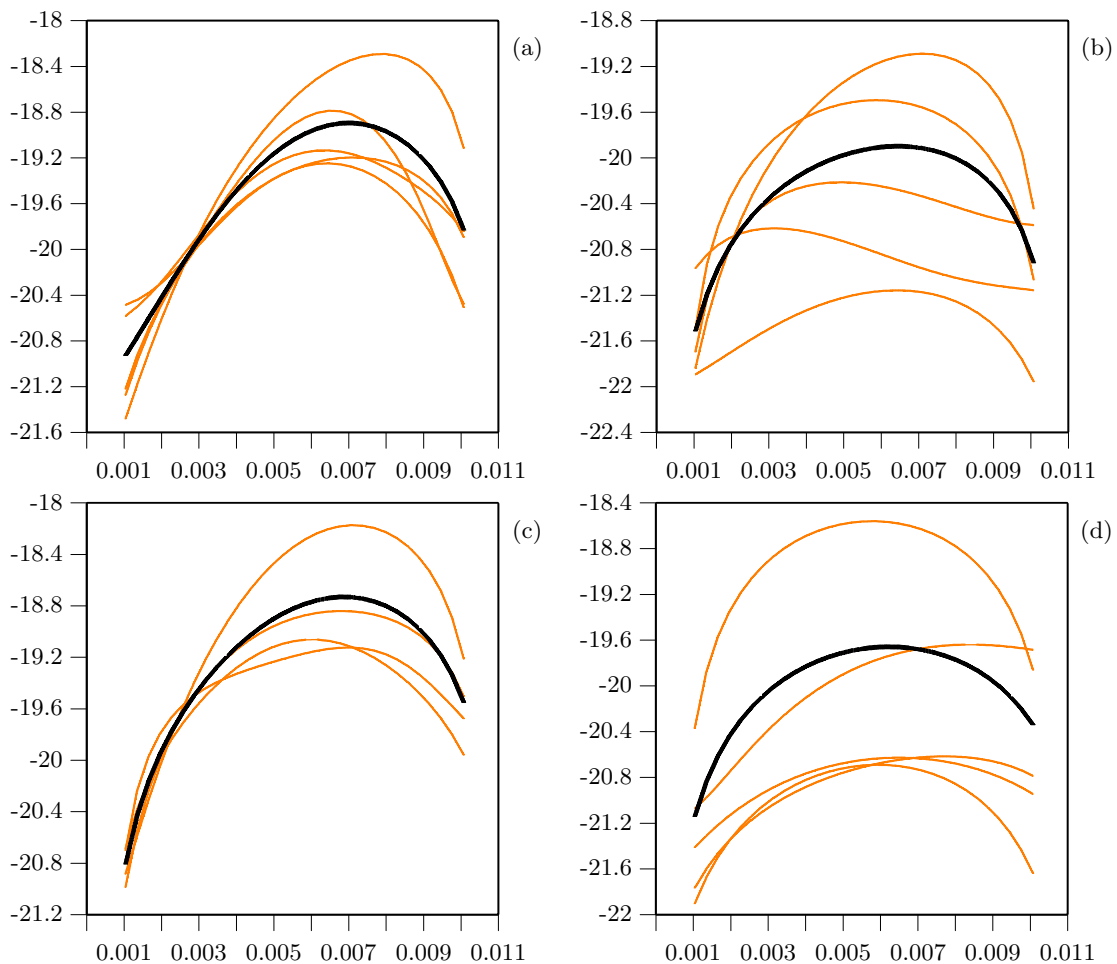


Figure 5.6.3. Courbes de vraisemblance pour l'exemple B: (a) et (c) méthode originale, (b) et (d) procédure limitant les recombinaisons. (a) et (b) 1M d'itérations pour chaque courbe et la courbe cumulée, (c) et (d) 5M d'itérations pour chaque courbe et la courbe cumulée.

Comme on peut le voir, cette modification ne semble pas apporter d'améliorations dans les estimations. Évidemment, le programme devient plus rapide puisque moins d'événements de recombinaison sont considérés.

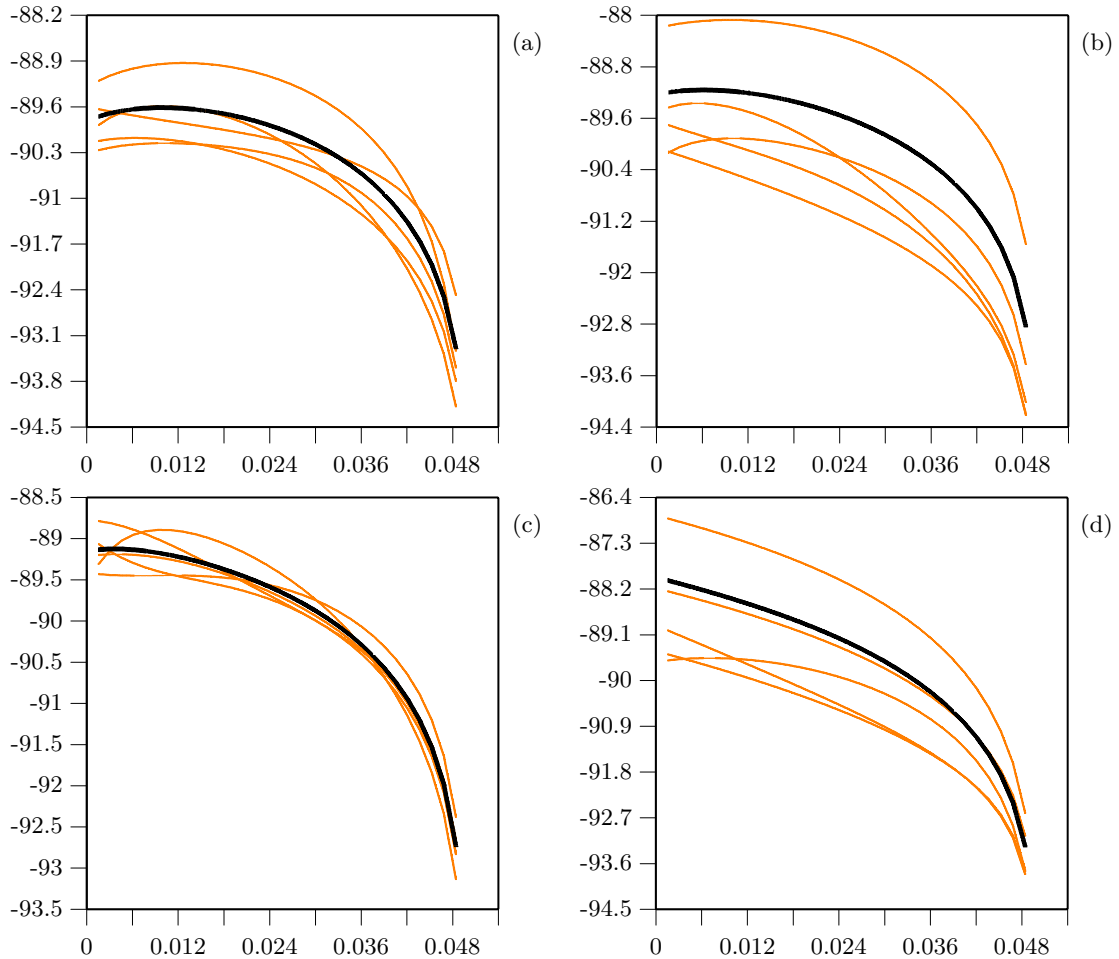


Figure 5.6.4. Courbes de vraisemblance pour l'exemple C: (a) et (c) méthode originale, (b) et (d) procédure limitant les recombinaisons. (a) et (b) 1M d'itérations pour chaque courbe et la courbe cumulée, (c) et (d) 5M d'itérations pour chaque courbe et la courbe cumulée.

5.7. Vers une meilleure distribution proposée

Dans le chapitre 3, nous avons utilisé l'équation de récurrence:

$$Q(\mathbf{H}_\tau) = \sum_{\substack{i, \\ n_i > 1}} \frac{S_{\mathbf{H}_\tau}}{D_{\mathbf{H}_\tau}} \frac{n(n_i - 1)}{S_{\mathbf{H}_\tau}} Q(\mathbf{H}_\tau + C_i)$$

$$\begin{aligned}
& + \sum_{\substack{i \neq j \\ \text{compatibles}}} \frac{S_{\mathbf{H}_\tau}}{D_{\mathbf{H}_\tau}} \frac{2n(n_k + 1 - \delta_{ik} - \delta_{jk})}{S_{\mathbf{H}_\tau}} Q(\mathbf{H}_\tau + C_{ij}^k) \\
& + \sum_i \sum_{\substack{m \in A_i, s_m^i \neq s_m^j \\ \text{pour tout } k \neq i}} \frac{S_{\mathbf{H}_\tau}}{D_{\mathbf{H}_\tau}} \frac{\theta_m(n_j + 1)}{S_{\mathbf{H}_\tau}} Q(\mathbf{H}_\tau + M_i^j(m)) \\
& + \sum_{i=1}^d \sum_{p=\gamma_i}^{\kappa_i} \frac{S_{\mathbf{H}_\tau}}{D_{\mathbf{H}_\tau}} \frac{\rho_p(n_j + 1)(n_k + 1)}{S_{\mathbf{H}_\tau}(n + 1)} Q(\mathbf{H}_\tau + R_i^{jk}(p)).
\end{aligned}$$

et la probabilité d'un événement de recombinaison $R_i^{jk}(p)$ dans l'intervalle p lors de la reconstruction du graphe était alors:

$$\frac{\rho_p(n_j + 1)(n_k + 1)}{S_{\mathbf{H}_\tau}(n + 1)}.$$

Supposons par exemple que nous ayons les trois séquences \mathbf{s}^1 , \mathbf{s}^2 et \mathbf{s}^3 suivantes, où les loci sont équidistants:

$$\mathbf{s}^1 \blacksquare \blacksquare \blacksquare \blacksquare \blacksquare \quad \mathbf{s}^2 \blacksquare \blacksquare \blacksquare \blacksquare \blacksquare \quad \mathbf{s}^3 \blacksquare \blacksquare \blacksquare \blacksquare \blacksquare$$

Alors, si une recombinaison se produit dans la séquence \mathbf{s}^2 , les différentes possibilités de parents sont:

$$\begin{aligned}
R_{\mathbf{s}^2}^{jk}(1) \quad \mathbf{s}^2 \blacksquare \blacksquare \blacksquare \blacksquare \blacksquare & \rightarrow j \blacksquare \blacksquare \blacksquare \blacksquare \blacksquare \quad k \blacksquare \blacksquare \blacksquare \blacksquare \blacksquare \\
R_{\mathbf{s}^2}^{jk}(2) \quad \mathbf{s}^2 \blacksquare \blacksquare \blacksquare \blacksquare \blacksquare & \rightarrow j \blacksquare \blacksquare \blacksquare \blacksquare \blacksquare \quad k \blacksquare \blacksquare \blacksquare \blacksquare \blacksquare \\
R_{\mathbf{s}^2}^{jk}(3) \quad \mathbf{s}^2 \blacksquare \blacksquare \blacksquare \blacksquare \blacksquare & \rightarrow j \blacksquare \blacksquare \blacksquare \blacksquare \blacksquare \quad k \blacksquare \blacksquare \blacksquare \blacksquare \blacksquare
\end{aligned}$$

Chacun de ces trois événements de recombinaison a alors la même probabilité d'être retenu pour reconstruire le graphe, car chaque paire de parents n'existe pas, i.e. on a $n_i = n_j = 0$. Un événement semble pourtant plus favorable que les autres. Si la recombinaison a lieu dans le deuxième intervalle, chacune des séquences parentales du recombinant va pouvoir coalescer avec les autres séquences déjà présentes dans l'échantillon (\mathbf{s}^1 et \mathbf{s}^3) sans recombinaison supplémentaire, comme l'illustre la figure 5.7.1.

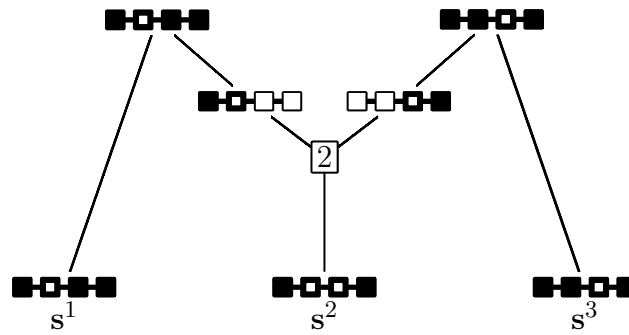


Figure 5.7.1. Une recombinaison dans le second intervalle de la séquence s^2 conduit à cette généalogie (voir texte).

En ce sens, l'information sur le matériel non ancestral peut brouiller les pistes, plutôt que de les éclairer. Nous allons donc considérer une simplification de l'arbre de recombinaison ancestral en ne tenant plus compte de la majorité du matériel non ancestral. Avec cette procédure simplifiée, quand une recombinaison se produit dans l'intervalle p de la séquence i , chacune des séquences compatibles à gauche (à droite) de p de la génération présente aurait pu être un parent "de gauche" ("de droite") de i . Par compatibles, on entend deux séquences qui peuvent coalescer. Ainsi, par exemple, la séquence s^1 de la figure 5.7.1 est un parent de gauche compatible avec la séquence s^2 si une recombinaison se produit dans les premier ou second intervalles; la séquence s^3 est un parent de droite compatible avec s^2 si une recombinaison se produit dans les second ou troisième intervalles.

Soient n'_j et n'_k le nombre de séquences présentes dans le graphe à un moment donné qui sont compatibles avec les parents de "gauche" et de "droite" de la séquence qui recombine. Notons que n'_j et n'_k dépendent de la séquence i et du point de recombinaison. Nous ne considérons dans les séquences compatibles que les séquences d'autres types, afin d'éviter des boucles de recombinaisons. La probabilité d'un événement de recombinaison est alors:

$$\frac{\rho_p(n'_j + 1)(n'_k + 1)}{S_{\mathbf{H}_\tau}(n + 1)},$$

et l'équation de récurrence reste semblable, simplement en remplaçant n_j par

n'_j et n_k par n'_k . Ainsi, la probabilité d'un événement de recombinaison tient compte du nombre de parents potentiels dans l'échantillon ancestral. Maintenant, il faut choisir adéquatement les parents de la séquence qui vient de recombiner. Chacun des parents sera choisi au hasard parmi les séquences compatibles en tenant compte du poids de chaque type de séquence dans l'échantillon ancestral. Si aucun parent compatible n'existe, alors le parent sera construit de la même façon qu'auparavant, en insérant du matériel non ancestral. Afin d'illustrer l'information supplémentaire sur les parents potentiels que nous utilisons, considérons le graphe de la figure 5.7.2. On observe qu'un simple événement de recombinaison s'est produit. Le tableau 5.7.3 montre, pour les quatre premières étapes du graphe (en partant de la base, nous essayons ici de le reconstruire), les valeurs des couples n_j/n_k et n'_j/n'_k pour chacun des types de séquence et tous les intervalles de recombinaison. Ainsi, on voit par exemple qu'à l'étape 3, si une recombinaison se produit dans le second intervalle, alors $n_j = 0/n_k = 0$ et $n'_j = 2/n'_k = 1$ pour la séquence s^9 . On constate bien que l'information sur les parents potentiels nous apporte une nouvelle information dont nous ne tenions pas compte auparavant.

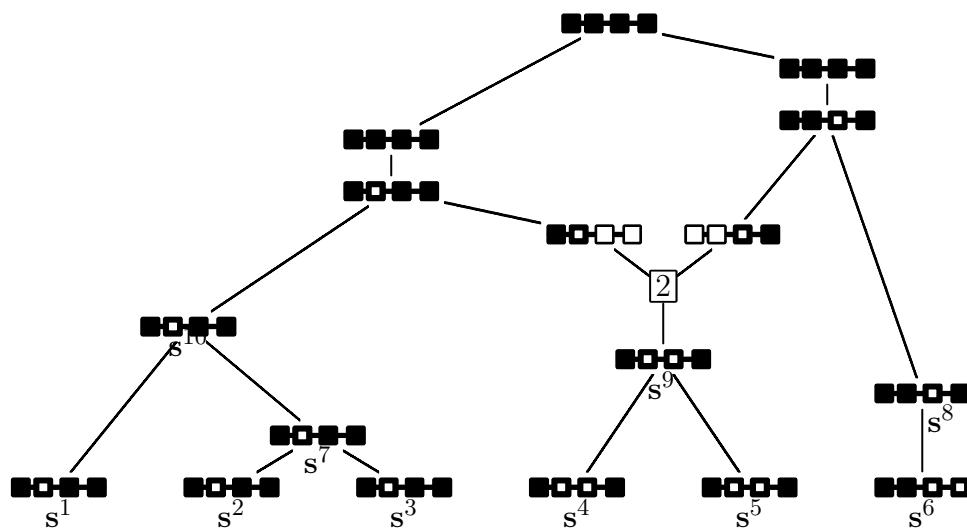


Figure 5.7.2. Illustration d'un graphe de recombinaison ancestral montrant un événement de recombinaison dans le second intervalle.

Tableau 5.7.3. Valeurs des couples n_j/n_k et n'_j/n'_k pour chacune des séquences de la figure 5.7.2, pour les quatre premières étapes de la reconstruction du graphe (- signifie 0).

Étape	Séquences	$R..(1)$		$R..(2)$		$R..(3)$	
		n_j/n_k	n'_j/n'_k	n_j/n_k	n'_j/n'_k	n_j/n_k	n'_j/n'_k
0	$s^1/s^2/s^3$	-/-	3/-	-/-	2/-	-/-	-/2
	s^4/s^5	-/-	4/-	-/-	3/-	-/-	-/3
	s^6	-/-	5/-	-/-	-/-	-/-	-/-
1	s^1/s^7	-/-	3/-	-/-	2/-	-/-	-/2
	s^4/s^5	-/-	3/-	-/-	2/-	-/-	-/2
	s^6	-/-	4/-	-/-	-/-	-/-	-/-
2	s^1/s^7	-/-	3/-	-/-	2/-	-/-	-/2
	s^4/s^5	-/-	3/-	-/-	2/1	-/-	-/3
	s^8	-/-	4/-	-/-	-/2	-/-	-/4
3	s^1/s^7	-/-	2/-	-/-	1/-	-/-	-/2
	s^9	-/-	3/-	-/-	2/1	-/-	-/3
	s^8	-/-	3/-	-/-	-/1	-/-	-/3

Pour illustrer que cette procédure propose de meilleurs choix de recombinaison, prenons l'exemple ci-dessus. Celui-ci est tout à fait hypothétique, et a été construit pour les besoins de l'illustration. En prenant les données:

n_i	Trait	Haplotype
3	1	0 0 0
2	1	0 1 0
1	0	0 1 1

et en cherchant r_T dans le premier intervalle entre marqueurs, nous nous retrouvons dans la situation de la figure 5.7.2. Si nous supposons que les deux intervalles entre marqueurs ont la même longueur et que nous regardons le nombre d'événements moyen nécessaire à la reconstruction du graphe, nous obtenons:

ρ	Nombre moyens d'événements	
	Procédure originale	Procédure simplifiée
3.2	22.8 ± 8.4	17.5 ± 5.1
6.4	24.6 ± 8.5	17.6 ± 5.1
12.8	28.3 ± 9.3	17.3 ± 5.0

Nous observons donc qu'en moyenne près de 23 événements sont nécessaires à la reconstruction du graphe pour $\rho = 3.2$ avec la procédure originale, alors que seulement 17 événements sont en moyenne nécessaires avec la procédure modifiée. Il est également intéressant de noter que plus ρ augmente, plus le nombre d'étapes nécessaire à la reconstruction du graphe grandit avec la procédure originale, alors qu'il est très stable avec la procédure simplifiée. Ceci est important, puisque plus un graphe est long en nombre d'événements, plus son poids sera petit. Les graphes les plus courts ont tendance à être les "meilleurs". Il faut cependant être prudent: le fait que les graphes soient plus courts n'est pas un gage de qualité de la distribution proposée.

D'un point de vue de calcul informatique, une complication se présente. Il nous faut calculer, à chaque étape du graphe et pour chaque événement de recombinaison, les nombres n'_j et n'_k . Plutôt que de simplement compter les séquences compatibles en comparant la séquence qui recombine avec les autres séquences présentes dans l'échantillon ancestral, nous avons modifié le programme pour que les séquences soient stockées dans des arbres. La figure 5.7.3 en montre une illustration, où les données correspondent à celles du graphe de la figure 5.7.2.

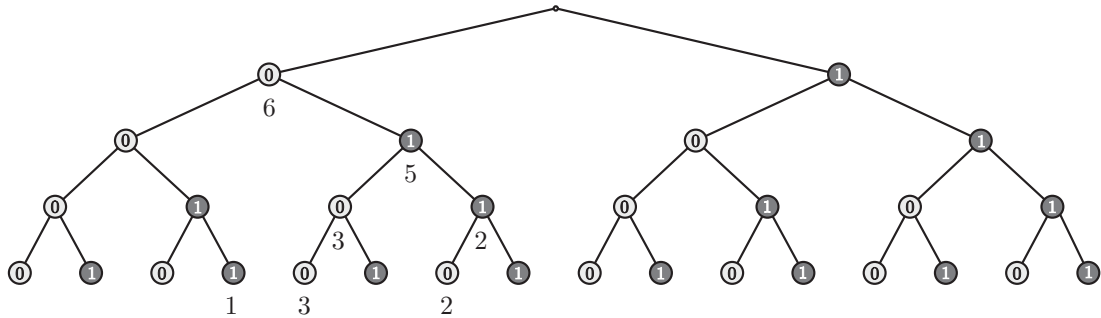


Figure 5.7.3. Illustration d'un arbre pour stocker l'information des haplotypes.

Ainsi, si l'on cherche les parents compatibles avec la séquence $(0, 1, 1, 0)$, nous n'avons seulement qu'à parcourir l'arbre, et nous voyons que si une recombinaison a lieu dans le premier intervalle, le nombre de parents partageant le même début de séquence (0) est de 6; si elle a lieu dans le second intervalle, le nombre de parents partageant le même début de séquence $(0, 1)$ est 5, et ainsi de suite.

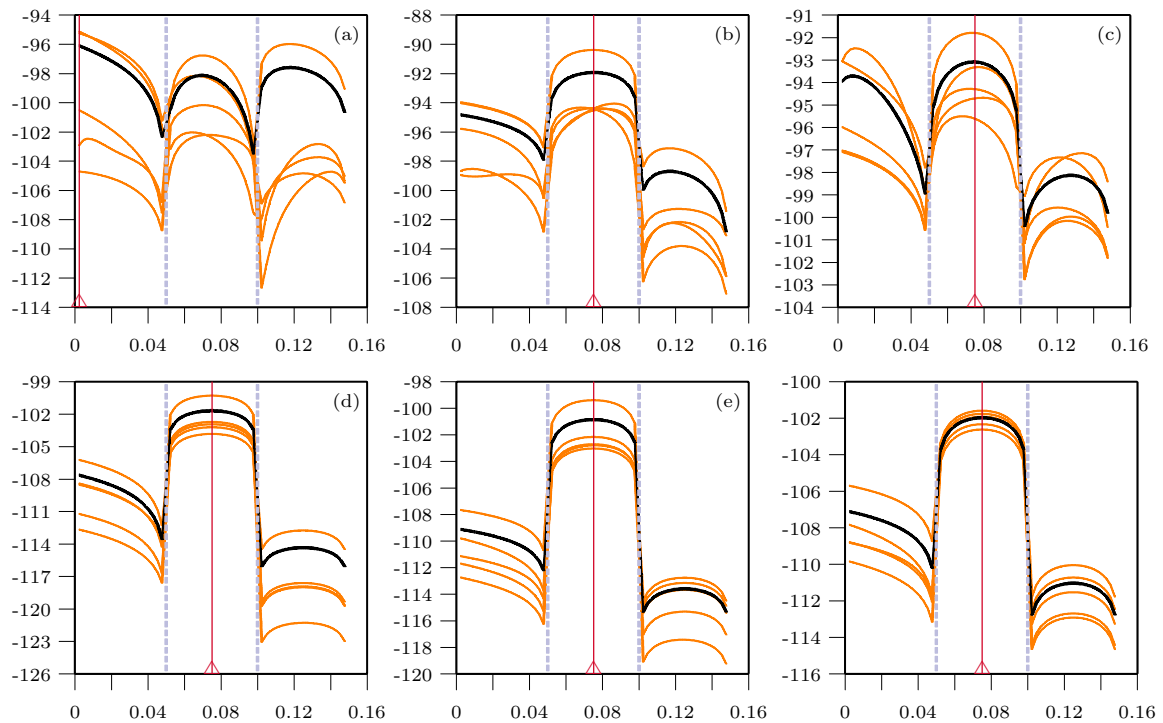


Figure 5.7.4. Courbes de vraisemblance pour l'exemple A: (a), (b) et (c) procédure originale; (d), (e) et (f) procédure simplifiée; (a) et (d) 16m et 10m d'itérations respectivement; (b) et (e) 160m et 100m d'itérations respectivement; (c) et (f) 1.6M et 1M d'itérations respectivement.

En fait, comme on a besoin des parents compatibles de gauche et de droite, l'arbre est plus complexe que celui illustré dans la figure 5.7.3: à chaque nœud, deux informations sont incorporées. L'arbre est mis à jour chaque fois qu'une modification est faite à l'échantillon ancestral; ainsi, si une séquence disparaît (par coalescence par exemple), on ajuste le contenu de l'arbre en parcourant les bons nœuds.

Nous avons effectué une série de simulations afin d'évaluer le comportement de cette procédure simplifiée. Nous utiliserons nos trois exemples habituels. Comme cette procédure simplifiée implique des calculs plus longs, nous en tenons compte dans les comparaisons. Ainsi, nous comparons les deux procédures à temps de calcul équivalent: pour l'ensemble de données A par exemple, pendant que

la

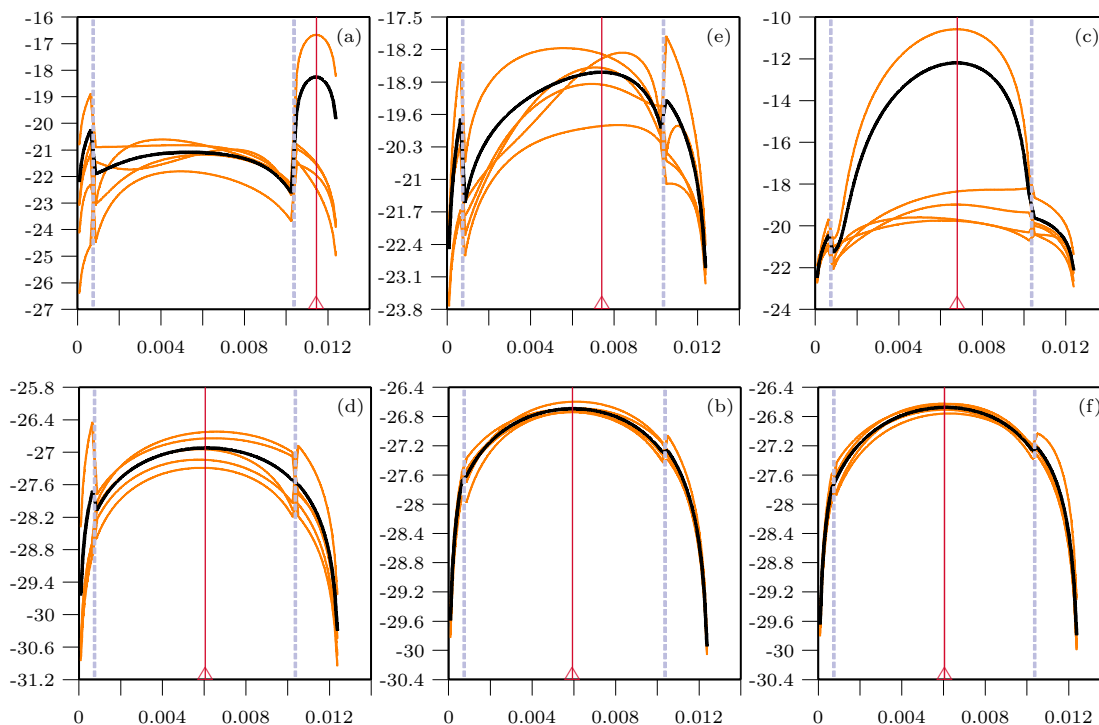


Figure 5.7.5. Courbes de vraisemblance pour l'exemple B: (a), (b) et (c) procédure originale; (d), (e) et (f) procédure simplifiée; (a) et (d) 16m et 10m d'itérations respectivement; (b) et (e) 160m et 100m d'itérations respectivement; (c) et (f) 1.6M et 1M d'itérations respectivement.

procédure originale effectuée 16m d'itérations, la procédure simplifiée n'en effectue que 10m; nous comparons donc 16m d'itérations de la procédure originale avec 10m d'itérations de la procédure simplifiée. La figure 5.7.4 montre les résultats des simulations pour l'exemple A. On peut voir que bien que le profil de la vraisemblance soit similaire pour les deux procédures, la procédure simplifiée montre une moins grande variation dans les profils de vraisemblance, et ce, même avec un faible nombre d'itérations. Avec la procédure simplifiée, les courbes sont très similaires quel que soit le nombre d'itérations utilisé.

Les simulations pour l'exemple B sont présentées dans la figure 5.7.5. La différence entre les deux procédures est flagrante. Les courbes de vraisemblance sont beaucoup moins variables avec la procédure simplifiée qu'avec l'autre. Notons cependant que l'estimation de r_T , bien que constante, est un peu moins bonne qu'avec la procédure originale.

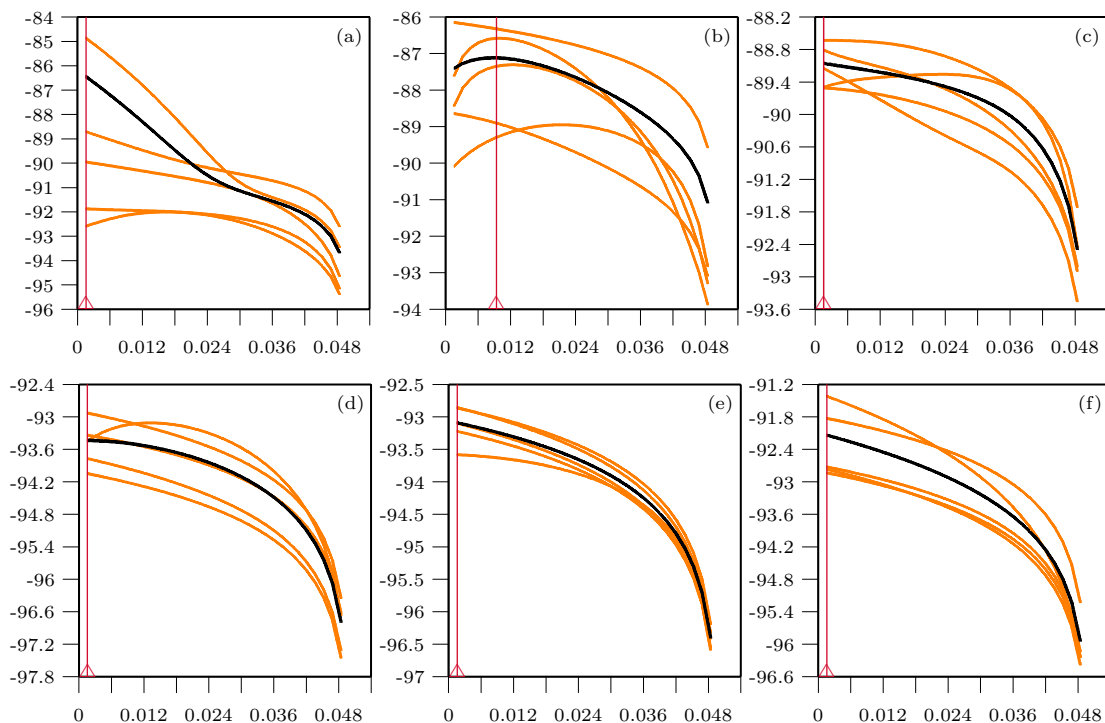


Figure 5.7.6. Courbes de vraisemblance pour l'exemple C: (a), (b) et (c) procédure originale; (d), (e) et (f) procédure simplifiée; (a) et (d) 17.5m et 10m d'itérations respectivement; (b) et (e) 175m et 100m d'itérations respectivement; (c) et (f) 1.75M et 1M d'itérations respectivement.

La figure 5.7.6 montre les simulations pour l'exemple C. On remarque immédiatement que la procédure simplifiée montre encore beaucoup moins de variation que la procédure originale. Notons que l'estimation de r_T est par contre moins bonne. La procédure simplifiée donne une estimation de r_T au tout début de la séquence, c'est-à-dire que le TIM serait en linkage avec le premier marqueur, alors que la vraie valeur de r_T est ici proche de 0.010. Cependant, cette estimation n'est pas facile, comme on peut le voir en examinant les estimations données par la procédure originale: celle-ci ne donne une estimation assez proche de 0.010 seulement qu'avec 100m d'itérations.

Afin de bien illustrer la différence entre les deux procédures, nous avons fait d'autres simulations sur les données de l'exemple C, mais en utilisant un plus grand nombre d'itérations: 8.75M pour la procédure originale, et 5M pour la procédure simplifiée. Les résultats, qui sont présentés dans la figure 5.7.7, nous

montre clairement ce que nous avons aperçu dans les exemples de cette section: la procédure simplifiée montre beaucoup moins de variation, mais son estimation semble moins précise. Il nous faut cependant tempérer cette remarque, car si l'on examine le biais d'estimation des deux procédures sur ce dernier exemple, il est en fait le même, car le TIM est en fait exactement entre le début de la séquence et l'estimation donnée par la procédure originale. De plus, une estimation ponctuelle n'a qu'un intérêt limité, l'intervalle de confiance est bien plus important; et il est presque identique dans les deux cas.

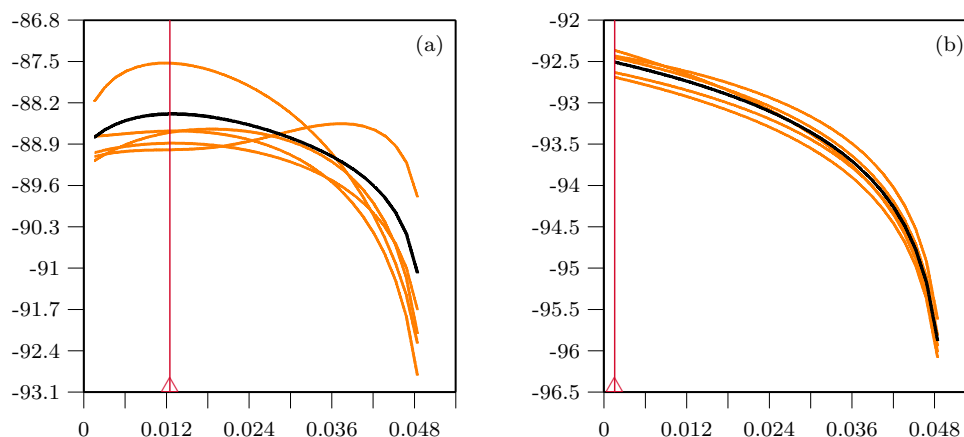


Figure 5.7.7. Courbes de vraisemblance pour l'exemple C: (a) procédure originale, 8.75M d'itérations; (b) procédure simplifiée, 5M d'itérations.

Nous avons vu dans cette section qu'un meilleur contrôle des événements de recombinaison pouvait améliorer les estimations de façon significative. Nous obtenons des estimations stables, même avec un petit nombre d'itérations. Pour cela, nous avons cependant dû faire quelques simplifications. La procédure simplifiée induit moins d'événements de recombinaison dans la reconstruction des graphes, ce qui fait que nous avons moins de variation dans les profils de vraisemblance. Cette procédure prend un certain sens si ρ est petit, car si la séquence est longue, la procédure simplifiée induira trop peu d'événements de recombinaison, et les estimations en résultant seront médiocres. De plus, les calculs sont plus complexes et plus longs; ils peuvent s'avérer difficiles à faire sur un grand ensemble de données réelles. Nous avons avec cette simplification une piste intéressante

pour améliorer la distribution proposée utilisée pour construire les graphes.

5.8. Rééchantillonnage

5.8.1. Idée originale de Liu

Chen et Liu (2000) et Liu (2001) ont proposé une méthode de rééchantillonnage dans le but d'améliorer la convergence de la méthode de Griffiths et Tavaré (1994c) dans le cas d'un processus de coalescence sans recombinaison avec mutation récurrente. Cette méthode est assez simple et semble, par l'exemple que les auteurs rapportent, fonctionner assez bien.

Soit $\mathcal{H} = \{H_{\tau^*}, \dots, H_1, H_0\}$ l'histoire complète de l'ensemble des séquences observées (H_0), où τ^* représente l'étape correspondant au début de la généalogie, c'est-à-dire à l'ancêtre commun le plus récent des séquences observées. Chaque H_τ est une liste non ordonnée des séquences ancestrales il y a τ générations. Soit $\theta = 4Nu$ le paramètre de mutation. La distribution de \mathcal{H} est alors donné par:

$$p_\theta(\mathcal{H}) \propto p_\theta(H_{\tau^*})p_\theta(H_{\tau^*-1} | H_{\tau^*}) \dots p_\theta(H_0 | H_1)p_\theta(\text{stop} | H_0),$$

où $p_\theta(\mathcal{H}_{\tau^*}) = \pi_0(\mathcal{H}_{\tau^*})$ et π_0 est la distribution stationnaire des probabilités de transition P_{ij} du processus de mutation. D'après Stephens et Donnelly (2000), on a:

$$p_\theta(H_{\tau-1} | H_\tau) = \begin{cases} \frac{n_i}{n} \frac{\theta}{n-1+\theta} P_{ij} & \text{si } H_{\tau-1} = H_\tau - i + j \\ \frac{n_i}{n} \frac{n-1}{n-1+\theta} & \text{si } H_{\tau-1} = H_\tau + i \end{cases}$$

pour $\tau = 1, \dots, \tau^*$, et le processus est arrêté juste avant qu'un nouveau type soit produit, c'est-à-dire:

$$p_\theta(\text{stop} | H_0) = \sum_i \frac{n_i}{n} \frac{n-1}{n-1+\theta},$$

où n est la taille de H_τ , et n_i est le nombre de séquences de type i dans H_τ . La notation $H_{\tau-1} = H_\tau + i$ indique que la nouvelle génération $H_{\tau-1}$ est formée

à partir de H_τ par la duplication d'une lignée de type i ; $H_{\tau-1} = H_\tau - i + j$ signifie que $H_{\tau-1}$ est obtenu de H_τ par une mutation d'un type i à un type j . Le calcul de la vraisemblance est ainsi fait en avant dans le temps, en partant de l'ancêtre commun, correspondant à l'état H_{τ^*} , jusqu'à l'échantillon des séquences observées, correspondant à l'état H_0 .

Dans une stratégie d'échantillonnage pondéré séquentielle, équivalente à la méthode de Griffiths et Tavaré (1994c), il est possible de simuler successivement H_1, H_2, \dots à partir de H_0 et d'une distribution proposée construite en inversant les probabilités "en avant" dans le temps pour un certain θ_0 fixé. Pour $\tau = 1, \dots, \tau^* - 1, \tau^*$, on définit:

$$g_{\theta_0}(H_\tau | H_{\tau-1}) = \frac{p_{\theta_0}(H_{\tau-1} | H_\tau)}{\sum_{H'_\tau} p_{\theta_0}(H_{\tau-1} | H'_\tau)},$$

et la distribution finale est:

$$g_{\theta_0}(\mathcal{H} | H_0) = g_{\theta_0}(H_1 | H_0) \cdots g_{\theta_0}(H_{\tau^*} | H_{\tau^*-1}) = \prod_{\tau=1}^{\tau^*} g_{\theta_0}(H_\tau | H_{\tau-1}).$$

En simulant m copies de cette distribution, on obtient m histoires $\mathcal{H}^{(1)}, \dots, \mathcal{H}^{(m)}$ à partir de H_0 , et la vraisemblance de H_0 s'écrit:

$$\begin{aligned} p_{\theta_0}(H_0) &\propto \sum_{H_1, \dots, H_{\tau^*}} \left[\prod_{\tau=1}^{\tau^*} p_{\theta_0}(H_{\tau-1} | H_\tau) \right] p_{\theta_0}(H_{\tau^*}) p_{\theta_0}(\text{stop} | H_0) \\ &\propto \sum_{H_1, \dots, H_{\tau^*}} \frac{\left[\prod_{\tau=1}^{\tau^*} p_{\theta_0}(H_{\tau-1} | H_\tau) \right]}{\left[\prod_{\tau=1}^{\tau^*} g_{\theta_0}(H_\tau | H_{\tau-1}) \right]} \left[\prod_{\tau=1}^{\tau^*} g_{\theta_0}(H_\tau | H_{\tau-1}) \right] p_{\theta_0}(H_{\tau^*}) \\ p_{\theta_0}(\text{stop} | H_0) &\propto E_{g_{\theta_0}} \left[\prod_{\tau=1}^{\tau^*} \frac{p_{\theta_0}(H_{\tau-1} | H_\tau)}{g_{\theta_0}(H_\tau | H_{\tau-1})} \right] p_{\theta_0}(H_{\tau^*}) p_{\theta_0}(\text{stop} | H_0) \\ &\propto E_{g_{\theta_0}} \left[\frac{p_{\theta_0}(\mathcal{H})}{g_{\theta_0}(\mathcal{H})} \right], \end{aligned}$$

et elle peut alors être estimée par:

$$\hat{p}_{\theta_0}(H_0) = \frac{1}{m} \sum_{j=1}^m \frac{p_{\theta_0}(\mathcal{H}^{(j)})}{g_{\theta_0}(\mathcal{H}^{(j)})}.$$

Le poids courant, pour $\tau < \tau^*$, d'une généalogie $\mathcal{H}^{(j)}$ est alors:

$$w_\tau = \frac{p_{\theta_0}(H_{\tau-1} | H_\tau) \cdots p_{\theta_0}(H_0 | H_1)}{g_{\theta_0}(H_\tau | H_{\tau-1}) \cdots g_{\theta_0}(H_1 | H_0)} \equiv w_{\tau-1} \frac{p_{\theta_0}(H_{\tau-1} | H_\tau)}{g_{\theta_0}(H_\tau | H_{\tau-1})}.$$

Le poids final de la généalogie $\mathcal{H}^{(j)}$ ($j = 1, \dots, m$) est $w = w_{\tau^*} p_{\theta_0}(H_{\tau^*}) \times p_{\theta_0}(stop | H_0)$. La méthode proposée consiste alors à construire m histoires ancestrales concurrentes à partir de H_0 . Tous les poids $w_\tau^{(1)}, \dots, w_\tau^{(m)}$ sont enregistrés à chaque étape τ . En construisant toutes les histoires jusqu'à ce que l'échantillon soit de taille n , le poids de l'histoire j est alors noté $w_{(j)}^{(n)}$; si le coefficient de variation des m poids excède un certain seuil, un rééchantillonnage est effectué, sinon, on poursuit les m histoires jusqu'à ce que les m échantillons arrivent à la taille $n+1$. En rééchantillonnant, on produit un nouvel échantillon de même taille en prenant un échantillon avec remise dans les m histoires, basé sur des probabilités proportionnelles à $\{w_{(n)}^{(1)}, \dots, w_{(n)}^{(m)}\}$. Le poids de chaque nouvelle histoire à cette étape est alors la moyenne des poids des m histoires avant rééchantillonnage. La figure 5.8.1 illustre par un exemple l'utilisation de la méthode pour l'estimation du paramètre θ .

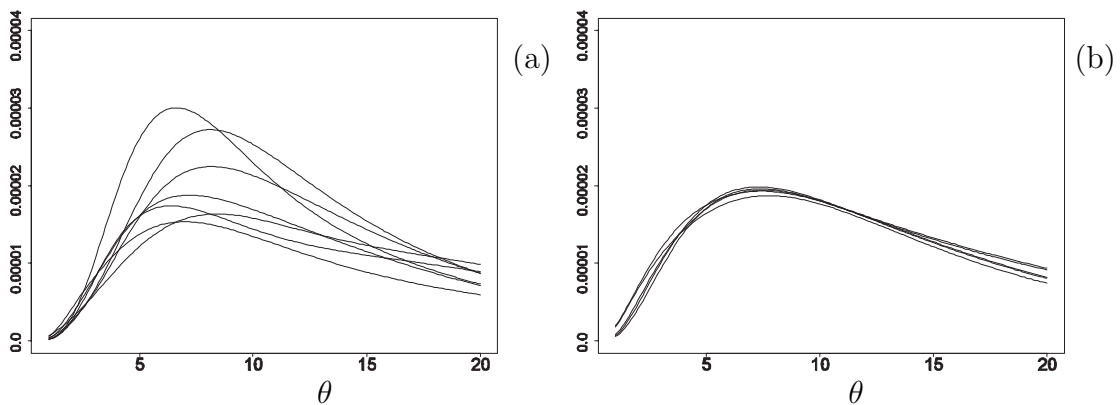


Figure 5.8.1. Tiré de Liu (2001). Exemple d'estimation de θ pour un processus de coalescence pur; (a) sept estimations indépendantes par la méthode de Griffiths et Tavaré (1994b); (b) cinq estimations indépendantes avec rééchantillonnage par la méthode de Liu (2001).

5.8.2. Adaptation au graphe de recombinaison ancestral

Afin de voir comment la technique du rééchantillonnage peut être adaptée à l'estimation des paramètres du graphe de recombinaison ancestral, nous allons commencer par présenter certains aspects du comportement de la méthode. Tout d'abord, notons que la méthodologie proposée par Liu (2001) effectue le rééchantillonnage quand une certaine taille de l'échantillon ancestral est atteinte. Dans un processus de coalescence sans recombinaison, la taille de l'échantillon diminue ou reste égale à chaque étape du graphe. Avec le graphe de recombinaison ancestral, nous avons un processus de naissance et de mort, et une même taille peut être atteinte à de multiples reprises. Il est donc clair que la seule solution consiste à effectuer un rééchantillonnage la première fois que l'échantillon ancestral atteint une certaine taille. Dans la suite du texte, à chaque fois que nous parlerons d'une taille de l'échantillon ancestral, nous voudrions dire la première fois que cette taille est atteinte.

Le rééchantillonnage supposant qu'il existe un certain lien entre le poids d'un graphe jusqu'à une étape donnée et son poids jusqu'à l'étape finale τ^* , il est intéressant de connaître ce lien; à cette fin, nous allons étudier la corrélation des poids à des étapes différentes. Soit \bar{w}_n , la moyenne des poids la première fois que l'échantillon ancestral atteint la taille n . La figure 5.8.2 présente la corrélation pour un millier de graphes entre \bar{w}_n et \bar{w}_1 , pour $n = 2, \dots, 17$, corrélation que nous noterons r_n^1 .

Notons qu'une transformation logarithmique a été effectuée sur \bar{w}_n pour que les coefficients de corrélation linéaire aient un sens, ceci compte tenu de leur distribution log-normale. On voit que le $r_{\frac{1}{2}}^1$ est assez élevé, mais que la liaison s'affaiblit très rapidement lorsque n augmente. Afin de mieux décrire cette liaison, nous avons évalué les corrélations r_n^1 sur 500m graphes de l'exemple D pour toutes les tailles échantillonnales, c'est-à-dire pour des valeurs de n allant du nombre de séquences observées dans l'échantillon jusqu'à 1. Nous avons également calculé les coefficients de variation CV_n (sur les poids non transformés), ainsi que les

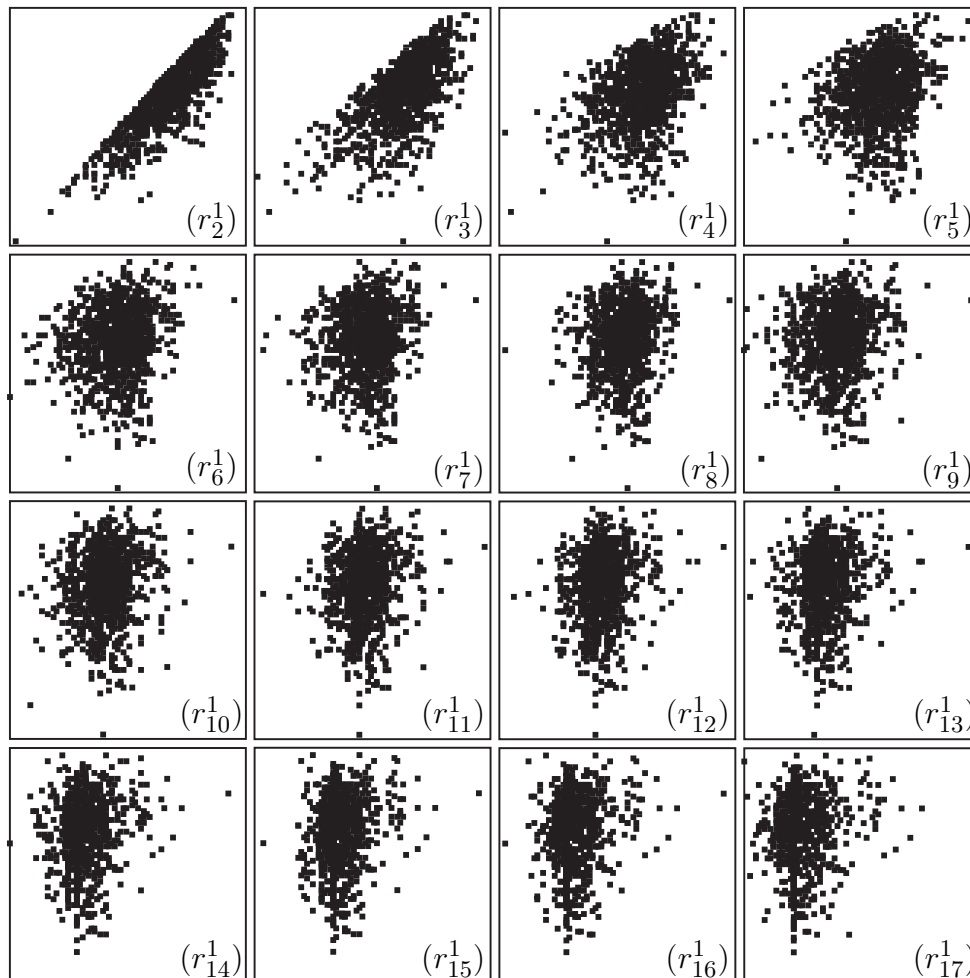


Figure 5.8.2. Corrélations r_n^1 entre \bar{w}_n et \bar{w}_1 , pour $n = 2, \dots, 17$, sur 1000 graphes de l'exemple D.

variances V_n à toutes ces étapes. Enfin, nous notons P_n la proportion du graphe reconstruit quand l'échantillon ancestral atteint la taille n pour la première fois; cette proportion est le nombre d'étapes nécessaires pour reconstruire le graphe jusqu'à cet instant divisé par le nombre total d'étapes nécessaires pour reconstruire le graphe en entier. La figure 5.8.3 présente ces différents calculs.

Comme on peut le voir, la corrélation des poids est pratiquement nulle jusqu'à la taille ancestrale 10, où elle commence lentement à augmenter; à $n = 2$, on a $r_2^1 = 0.17$, ce qui reste faible. Il est frappant de remarquer qu'à $n = 2$, la proportion moyenne de la généalogie construite n'est que d'environ 60%, ce qui

veut dire qu'il reste 40% à faire. On peut observer que la variance augmente très lentement, et le coefficient de variation commence à augmenter vers $n = 20$. Dans cet exemple, comme on l'a vu à la section 3.8, deux haplotypes majeurs sont présents: ils ont une taille respective de 65 et de 10, et représentent respectivement 74% et 13% de l'échantillon. Typiquement, nous avons beaucoup d'événements de coalescence identiques en début de généalogie; ces événements ayant une forte probabilité, il y a alors peu de variation. Notons que la taille échantillonnale de 88 moins l'effectif des deux haplotypes majeurs est 13, ce qui correspond au début de la variation des poids (voir 5.8.3, (c)).

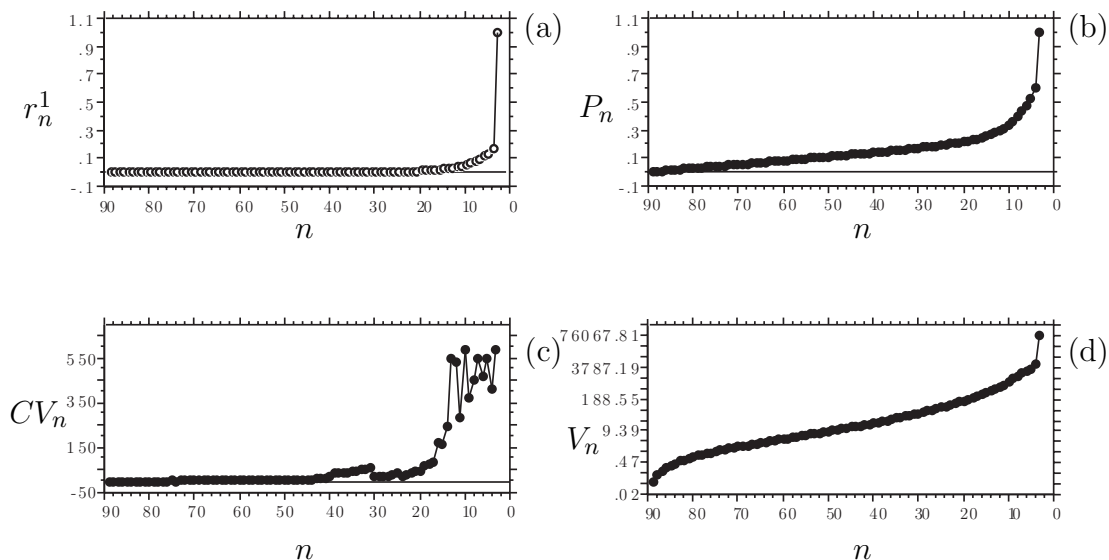


Figure 5.8.3. Évolution de la distribution et de la corrélation des poids pendant la reconstruction du graphe, à partir des données de l'exemple D.

La figure 5.8.4 présente des résultats similaires pour l'exemple B. Ici, l'haplotype majeur a un effectif de 5, ce qui représente 50% de l'échantillon. On remarque que la corrélation et la variance augmentent à partir de cette taille $n = 5$. Pour cet exemple, la moitié de la généalogie est construite vers $n = 3$ ou $n = 4$.

L'analyse des corrélations pour le dernier exemple (C), est présentée dans la figure 5.8.5. On a le même schéma qu'avec les autres exemples. Ici, nous avons un haplotype majeur de taille 78, représentant 78% de l'échantillon. À $n = 100 - 78 = 22$, on peut clairement voir à l'aide des coefficients de variation que c'est

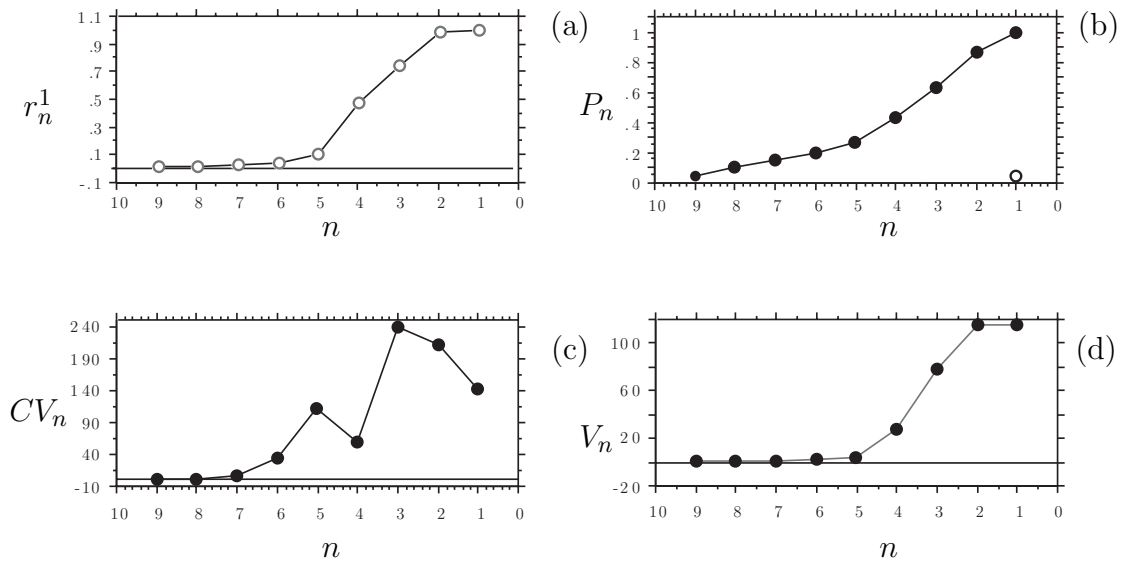


Figure 5.8.4. Évolution de la distribution et de la corrélation des poids pendant la construction du graphe à partir des données de l'exemple B.

précisément à cette taille échantillonnale qu'il commence réellement à y avoir variation. Par contre, ici comme dans l'exemple D que nous avons vu, la corrélation n'augmente que très tard: il faut atteindre $n = 6$ pour obtenir des corrélations qui peuvent être utiles pour faire du rééchantillonnage.

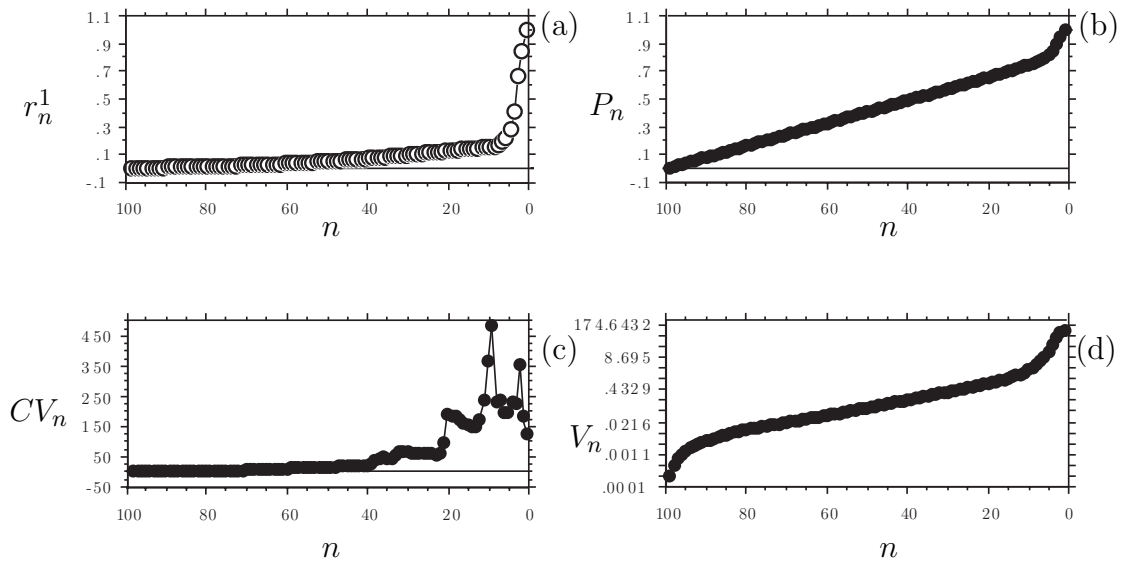


Figure 5.8.5. Évolution de la distribution et de la corrélation des poids pendant la construction du graphe à partir des données de l'exemple C.

Ces exemples nous montrent que l'on a bien corrélation entre les poids pendant la construction et les poids finaux, mais que celle-ci ne commence à être importante que très tard dans la construction du graphe. Notons cependant qu'à ce point-là, une part importante du graphe peut rester à construire; le rééchantillonnage peut quand même être alors un atout. Nous avons vu également que, de façon générale, nous commençons à avoir une variation dans les poids à partir de:

$$n - \sum_{\text{haplotypes majeurs}} n_i,$$

où n est la taille de l'échantillon au temps 0, n_i est la taille de l'haplotype i , et les haplotypes majeurs sont définis comme étant un (ou plusieurs) type(s) de séquences prédominantes dans l'échantillon; il s'agit souvent de types de séquences proches de l'haplotype ancestral.

Nous avons vu que Liu (2001) construit de multiples arbres de coalescence en même temps, et procède à un rééchantillonnage quand la variation des poids atteint un certain seuil. Il est malheureusement difficile de faire la même chose dans notre cas; l'information que l'on devrait stocker prendrait un grand espace mémoire, et compte tenu du grand nombre de graphes qu'il est nécessaire de construire, cela rend l'utilisation presque impossible. De plus, le temps de calcul en serait grandement allongé.

Nous allons maintenant présenter la méthodologie que nous utiliserons ici pour évaluer si le rééchantillonnage est une alternative viable pour nos estimations. Nous allons échantillonner m histoires ancestrales, jusqu'à ce que celles-ci atteignent la taille ancestrale n_S . Nous ne pouvons garder en mémoire qu'un certain nombre des m histoires ancestrales, nombre que nous noterons m' . À ce moment là, nous gardons alors les m' histoires ayant les poids les plus élevés parmi les m . Notons que si m n'est pas trop grand, c'est-à-dire ne dépasse pas un million, nous pouvons garder toutes les histoires échantillonnées. Au fur et à mesure que nous construisons les m échantillons, dès que nous arrivons à la

taille ancestrale n_S pour un échantillon donné, nous notons le poids du graphe, et nous avons ainsi un vecteur des poids

$$\mathbf{W} = (w^{(1)}, \dots, w^{(m')})$$

où $w^{(j)}$ ($1 \leq j \leq m'$) est le poids du graphe j à une certaine taille ancestrale choisie. Si $m' < m$, nous gardons les m' graphes ayant les poids les plus élevés. Nous verrons qu'en fait, même si m' est très inférieur à m , cela ne fait aucune différence dans les estimations, puisque les graphes ayant les plus petits poids n'ont presque aucune chance d'être rééchantillonnés. Dans le même temps que nous gardons le poids de chacun des m' graphes, nous gardons aussi la valeur du générateur aléatoire ayant servi à reconstruire chacun de ces graphes; nous réutilisons ces valeurs pour la phase de rééchantillonnage. En pratique, un graphe est entièrement déterminé par la valeur "racine" du générateur aléatoire. Ainsi, nous n'avons que peu d'informations à garder. Cependant, en échantillonnant, si $m' < m$, il faut que l'on trie \mathbf{W} , le vecteur des m' poids, à chaque graphe dont le poids est supérieur au plus petit élément du vecteur des poids. En effet, au fur et à mesure que nous construisons les m graphes, le poids des m' premiers graphes que nous reconstruisons sont entrés dans \mathbf{W} , puis quand le $(m' + 1)^{\text{ième}}$ graphe est reconstruit, il faut regarder si son poids est supérieur au plus petit élément de \mathbf{W} ; s'il est plus petit, on remplace le plus petit poids de \mathbf{W} par le poids du $(m' + 1)^{\text{ième}}$, s'il est plus grand ou égal au minimum de \mathbf{W} , on passe simplement à la reconstruction du $(m' + 2)^{\text{ième}}$ graphe, et ainsi de suite. La procédure se poursuit de la même façon pour les prochains graphes, ainsi, \mathbf{W} contient toujours les m' poids les plus élevés parmi les m .

Pour la phase de rééchantillonnage, nous prenons un échantillon de taille m avec remise des m' graphes, en sélectionnant les histoires proportionnellement à leur poids. Dans la pratique, afin de minimiser le temps de calcul, nous procédons légèrement différemment: au début de la phase de rééchantillonnage, nous déterminons le nombre de fois que nous prendrons le graphe j ($1 \leq j \leq m'$), celui-ci ayant une probabilité p_j d'être choisi, tel que:

$$p_j = \frac{w^{(j)}}{\sum_{i=1}^{m'} w^{(i)}},$$

et sachant que l'estimation est basée sur m graphes, le graphe j sera tiré en moyenne mp_j fois. En imposant la racine du générateur aléatoire, nous reconstruisons le graphe j jusqu'à la taille n_S , puis arrivé à n_S , nous notons en mémoire l'état du graphe: nombre de séquences, types des séquences, variables comme a , b , etc. La moyenne des poids de la phase d'échantillonnage est alors donnée à chaque graphe, ceci afin d'avoir une estimation correcte de la vraisemblance. À chaque graphe nous déroutons le générateur aléatoire (de façon aléatoire), et reconstruisons le graphe une première fois. La seconde fois que nous voulons rééchantillonner le même graphe j , nous repartons de la taille n_S , c'est-à-dire que nous n'avons plus à reconstruire les premières étapes menant à la taille n_S . Nous gagnons ainsi beaucoup de temps de calcul.

5.8.3. Exemple

Afin d'illustrer la procédure de rééchantillonnage, nous présentons ici un exemple avec les données A. Deux échantillonnages ont été effectués, chacun basé sur 50m d'itérations. Nous avons choisi $m' = 50m$, c'est-à-dire que nous gardons tous les graphes construits. De plus, nous avons choisi $n_S = 5$. Chaque échantillonnage a nécessité 40 minutes de calcul. Pour chacun des deux échantillonnages, deux rééchantillonnages ont été effectués: parmi les quatre courbes de vraisemblance obtenues, elles sont donc deux à deux dépendantes du même échantillonnage. Cette façon de faire nous permet de voir si la procédure est reproductible (les quatre courbes devraient être similaires), et si le rééchantillonnage ne "dépend pas trop" de l'échantillonnage, comme nous le verrons dans la suite par un exemple. Chacun des rééchantillonnages a exigé 20 minutes de calcul environ. Les quatre profils de vraisemblance sont présentés dans la figure 5.8.6. Compte tenu du faible nombre d'itérations, le résultat est assez intéressant. Les profils de vraisemblance sont similaires aux profils que nous avons déjà obtenus dans les autres analyses.

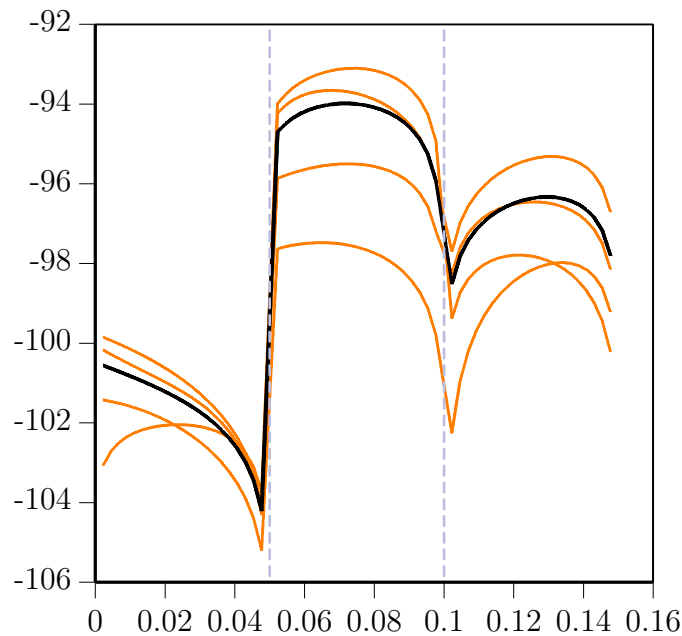


Figure 5.8.6. Deux échantillonnages de 50m d'itérations, avec $m' = 50m$, $n_S = 5$, et deux rééchantillonnages de 50m d'itérations pour chacun des échantillonnages. La courbe en gras est la courbe cumulée.

5.8.4. Effet du nombre de graphes utilisés pour le rééchantillonnage

Dans cette section, nous étudierons l'effet du choix de m' . Comme nous l'avons indiqué, nous ne pouvons, de façon générale, garder tous les poids et toutes les racines aléatoires des différents graphes construits. La figure 5.8.7 illustre le choix de m' avec l'exemple B. À partir d'un échantillon de 50m de graphes, sept rééchantillonnages ont été effectués, avec différentes valeurs de m' : 50m, 40m, 30m, 20m, 10m, 5m et 1m. Cette expérience a été répétée trois fois, pour des valeurs de n_S de 3, 4 et 5 (la taille de l'échantillon est 10 pour l'exemple B). Pour $n_S = 2$, les sept profils de vraisemblances sont quasi identiques. On commence à avoir de la variation pour $n_S = 4$ et surtout pour $n_S = 5$. La variation que l'on observe alors n'est pas forcément due à l'effet de m' , mais de n_S , comme nous le verrons par la suite.

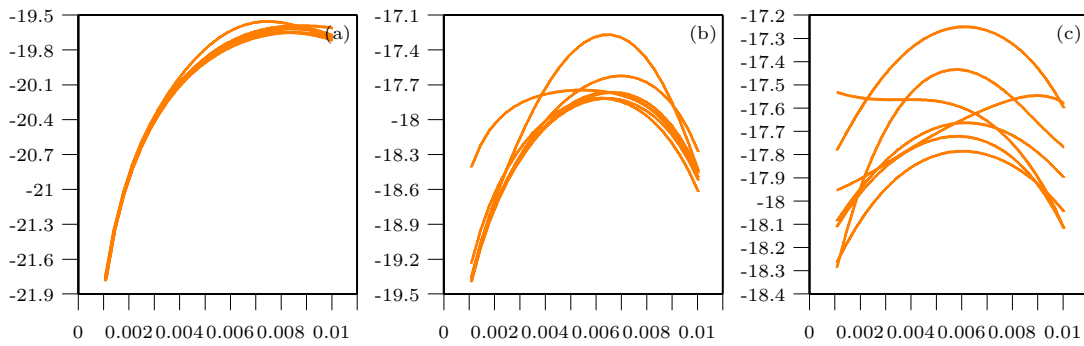


Figure 5.8.7. À partir d'un échantillon de 50m de graphes de l'exemple B, rééchantillonnage basé sur des valeurs de m' de 50m, 40m, 30m, 20m, 10m, 5m et 1m: (a) $n_S = 3$, (b) $n_S = 4$, (c) $n_S = 5$.

Une analyse similaire a été effectuée sur l'exemple C. Pour les trois tailles échantillonnales considérées, on voit que m' ne change presque rien. Quel que soit le nombre de graphe utilisé dans la phase de rééchantillonnage, nous obtenons un profil de vraisemblance similaire.

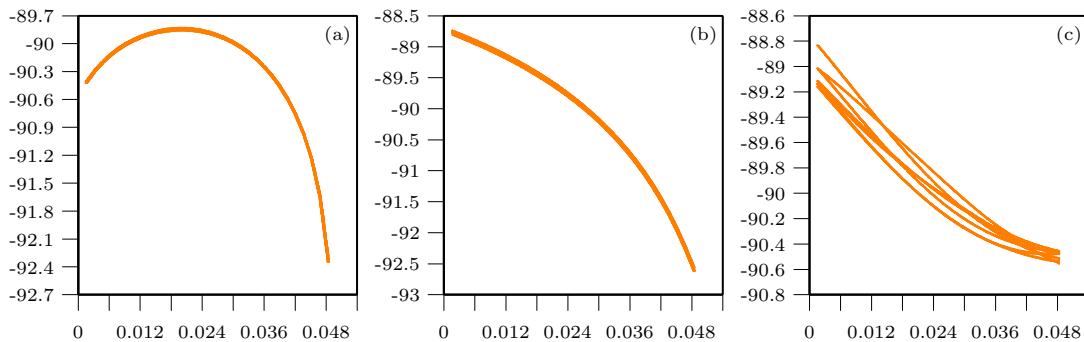


Figure 5.8.8. À partir d'un échantillon de 50m de graphes de l'exemple C, rééchantillonnage basé sur des valeurs de m' de 50m, 40m, 30m, 20m, 10m, 5m et 1m: (a) $n_S = 2$, (b) $n_S = 4$, (c) $n_S = 6$.

Dans la phase de rééchantillonnage, les graphes sont sélectionnés en fonction de leur poids; or, on sait que peu de graphes contribuent réellement à la vraisemblance. Donc dans notre échantillon de 50m de graphes, la probabilité de choisir une grande partie d'entre eux est presque nulle; c'est pourquoi le rééchantillonnage basé sur $m' = 50m$ de graphes nous donne un résultat très

similaire à un rééchantillonnage basé sur le millier de graphes ayant les poids les plus élevés parmi les 50m.

5.8.5. Tests de rééchantillonnage

Afin d'évaluer le rééchantillonnage, nous avons fait des tests sur les exemples B et C, en appliquant le même schéma que pour l'exemple A de la section 5.7.3: deux échantillons de tailles égales, puis, pour chacun d'eux, deux rééchantillonnages. Plusieurs n_S ont été utilisés. Les résultats sont présentés sur la figure 5.8.9 pour l'exemple B, et sur la figure 5.8.10 pour l'exemple C.

Comme on peut l'observer, les profils sont très variables. Les profils provenant d'un même échantillonnage tendent à être plus similaires que les profils pris dans leur ensemble.

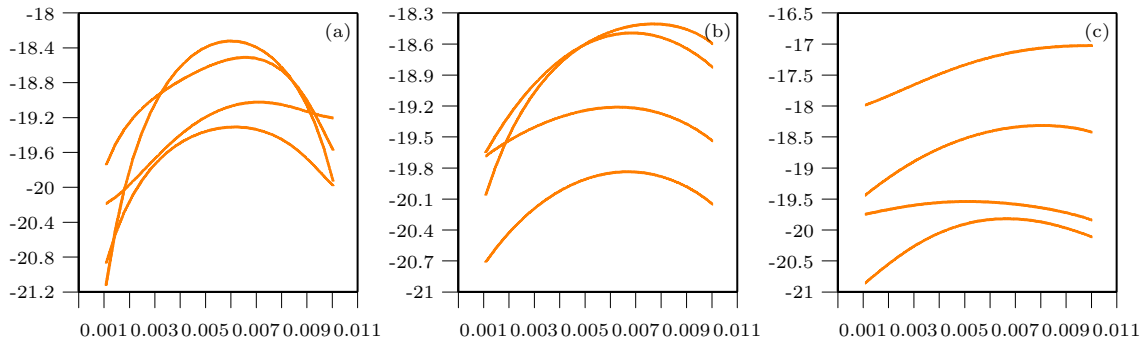


Figure 5.8.9. Échantillonnage et rééchantillonnage de 1M d'itérations pour l'exemple B: (a) $n_S = 2$, (b) $n_S = 3$, (c) $n_S = 4$ ($m' = 1m$).

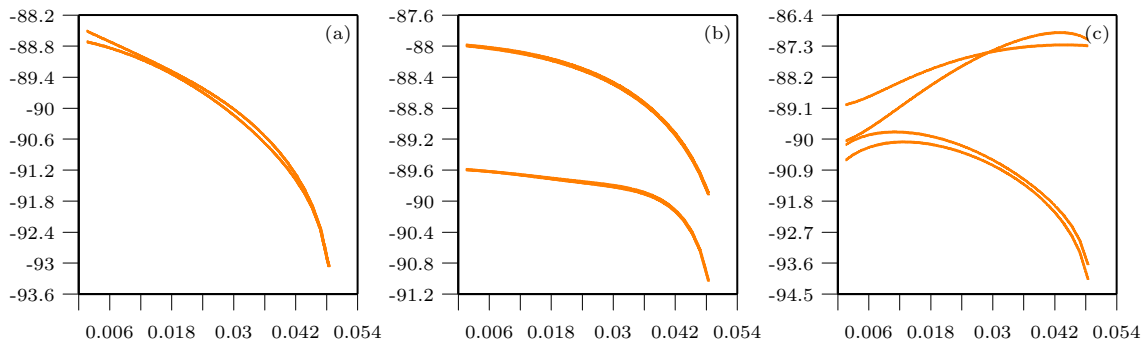


Figure 5.8.10. Échantillonnage et rééchantillonnage de 500m d'itérations pour l'exemple C: (a) $n_S = 2$, (b) $n_S = 3$, (c) $n_S = 4$ ($m' = 5m$).

Plusieurs autres simulations ont été effectuées en changeant le nombre d'itérations, m' , n_S , mais elles ne changent pas l'observation que l'on peut faire ici avec ces exemples. Rééchantillonner dans le graphe de recombinaison ancestral semble moins simple que dans un processus de coalescence sans recombinaison. Une raison est peut-être que beaucoup d'événements importants se passent à la fin de la généalogie, et les échantillons ainsi générés ne sont pas très bons. Cette méthodologie ne semble donc pas être une façon d'améliorer les inférences pour faire de la cartographie génétique fine par le graphe de recombinaison ancestral avec un modèle à mutations non récurrentes. Par contre, nous pouvons essayer de faire du rééchantillonnage avec un modèle de microsatellites. Nous avons plus de

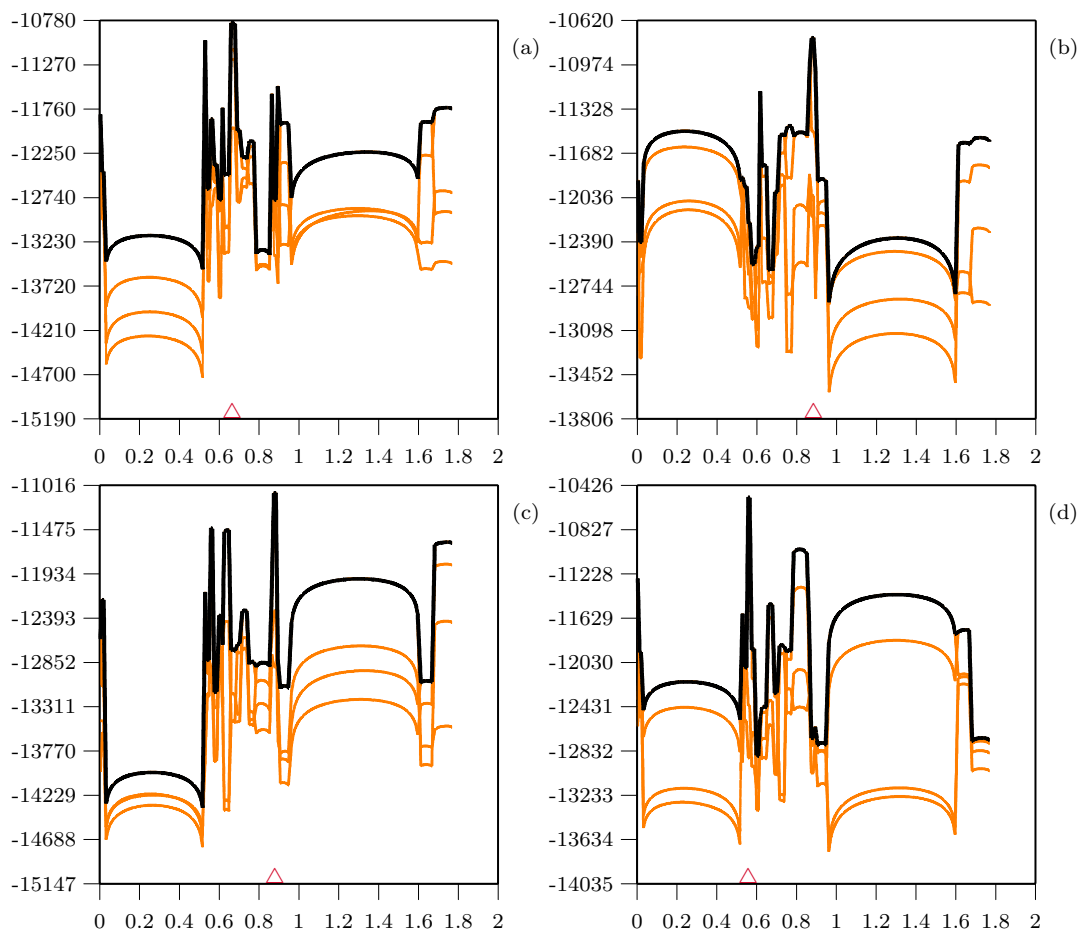


Figure 5.8.11. Rééchantillonnage pour l'exemple E ($m' = 1m$): (a) et (b) $n_S = 18$; (c) et (d) $n_S = 9$; (a) et (c) $m = 10m$; (b) et (d) $m = 25m$. Les courbes noires sont les courbes cumulées.

chance que cela fonctionne, puisque les mutations étant plus fréquentes, il y a plus de variation, plus tôt dans les graphes.

Nous avons donc fait des simulations sur les données de l'exemple E. La figure 5.8.11 montre des simulations de 10m d'itérations, (a) et (c), de 25m d'itérations, (b) et (d), pour différentes valeurs de n_s : 18 pour (a) et (b), et 9 pour (c) et (d). Notons que tous les cas sont inclus dans les analyses, y compris ceux qui ne possèdent pas la mutation $\Delta F508$. Nous obtenons des courbes de vraisemblance similaires à celles que nous avons obtenues dans la section 5.3. Une région centrale montre plus de déséquilibre de liaison, et nous avons toujours un pic de vraisemblance très proche de 0.88, la position exacte de $\Delta F508$. Cependant, il

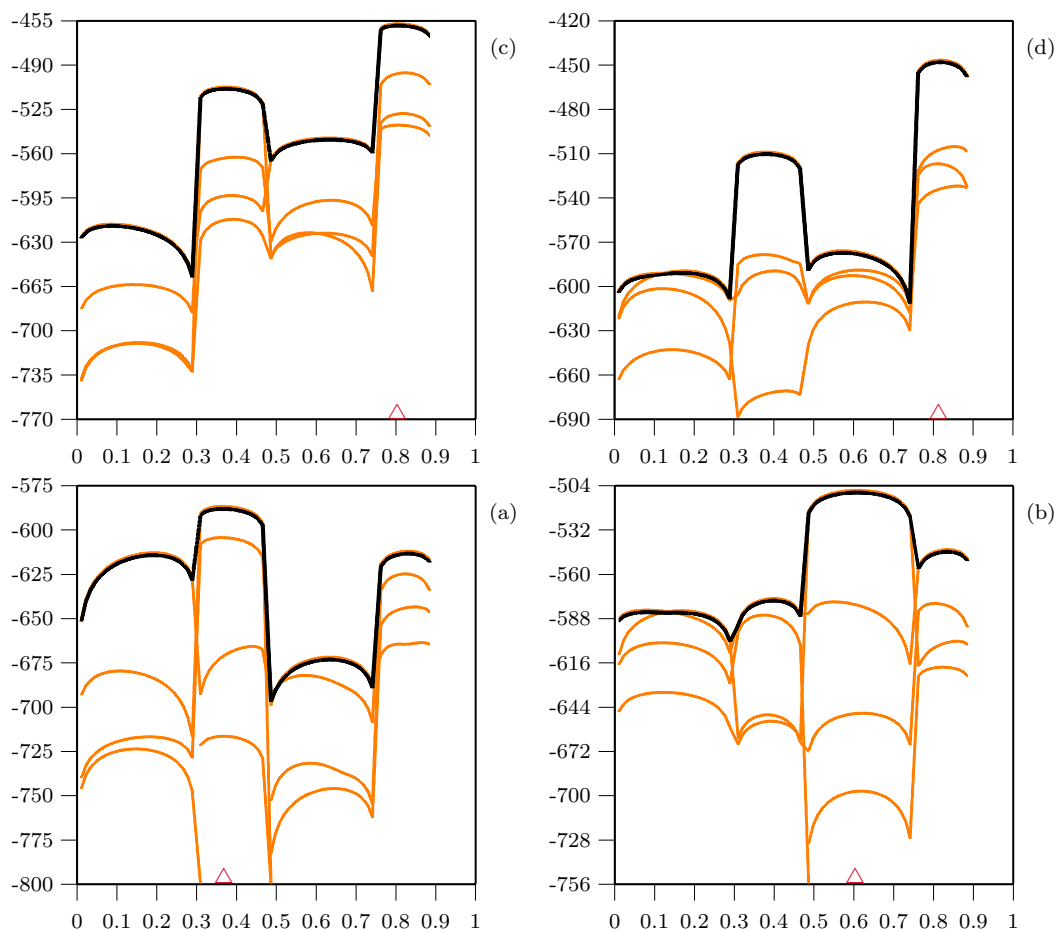


Figure 5.8.12. Rééchantillonnage pour l'exemple D ($m' = 1m$): (a) et (b) $n_s = 5$; (c) et (d) $n_s = 10$; (a) et (c) $m = 100m$; (b) et (d) $m = 250m$. Les courbes noires sont les courbes cumulées.

est évident que les courbes de vraisemblance varient beaucoup. Peu d'itérations sont ici utilisées, mais cela prend quand même, par exemple, deux jours et demi de calcul pour obtenir les quatre courbes de 250m d'itérations.

Enfin, nous avons utilisé le rééchantillonnage sur les données de l'exemple D. La figure 5.8.12 montre des simulations de 100m d'itérations, (a) et (c), de 250m d'itérations, (b) et (d), avec $n_S = 5$, (a) et (b), et $n_S = 10$, (c) et (d). Mis à part la simulation de 250m d'itérations avec $n_S = 5$, les simulations montrent un profil similaire aux résultats que nous avons obtenus avec un modèle à mutation non récurrente: la vraisemblance augmente dans le second intervalle, où la mutation est, mais le maximum de vraisemblance est tantôt dans le second intervalle, tantôt dans le quatrième. La variation des courbes de vraisemblance est tellement grande pour cet exemple, qu'il est même difficile d'interpréter les résultats. Comme nous l'avons vu, la séquence est probablement trop grande, et nous manquons d'informations pour pouvoir faire des estimations.

Avec l'étude de l'exemple E, nous voyons que le rééchantillonnage fonctionne. Celui-ci dépend cependant de la procédure d'estimation sous-jacente, qui elle montre beaucoup de variabilité. Peu d'itérations ont été utilisées, il est donc difficile de dire si le rééchantillonnage stabilise les courbes de vraisemblance. Aussi, nous ne faisons qu'une seule étape de rééchantillonnage. Nous gagnerions peut-être comme Liu (2001) à insérer plusieurs étapes de rééchantillonnage dans la reconstruction des graphes, mais il faudrait modifier le programme pour cela.

Conclusion

Le modèle décrivant l’histoire des séquences observées est un élément-clé d’une méthode de cartographie fine utilisant le déséquilibre de liaison. Le graphe de recombinaison apparaît comme un outil de choix pour modéliser l’histoire de telles séquences: il nous permet de tenir compte de la dépendance entre marqueurs et de la dépendance entre les séquences. De plus, le processus de coalescence a été peu utilisé à des fins de cartographie, et l’ARG ne l’a jamais été. Cela devenait alors un champ de recherche prometteur à explorer.

Nous avons alors introduit le modèle que nous proposons, par la définition d’une séquence ordonnée de marqueurs génétiques, dont l’un d’entre eux est à une position inconnue le long de la séquence, le TIM. Nous avons supposé que les mutations étaient non récurrentes et que la taille de population était constante. Nous avons alors défini les probabilités des événements de coalescence, de mutation et de recombinaison, et montré comment définir une équation de récurrence pour relier la probabilité d’une configuration de séquences à une autre dans le passé. Nous avons aussi montré comment définir un algorithme de Monte Carlo afin d’obtenir la vraisemblance de la position du TIM, et comment définir un schéma d’échantillonnage pondéré pour obtenir la vraisemblance à l’intérieur d’un intervalle pour d’autres valeurs que la valeur conductrice. L’effet des valeurs conductrices utilisées pour la reconstruction des graphes a été exploré; il semble qu’utiliser une seule valeur conductrice ne pose pas de problèmes. Les résultats montrent que nous obtenons une bonne estimation de r_T , même lorsque la vraie position du TIM est loin de la valeur conductrice. Une procédure pouvant nous

permettre de gagner du temps en choisissant d'abord une séquence, puis un intervalle de recombinaison a été définie et explorée. Ce type de procédure se révèle d'une grande importance dans les cas où nous disposons d'un grand nombre de marqueurs observés, et/ou d'un grand nombre de types de séquences.

La méthode a été évaluée à l'aide de trois exemples. Pour chacun d'eux, la méthode trouve adéquatement la position du TIM, bien qu'une variation des courbes de vraisemblance soit observée. Nous avons constaté que plus le nombre d'itérations utilisées pour évaluer la vraisemblance augmente, plus la variation semble diminuer, mais que ce phénomène est très lent. Aussi, la répétition de simulations avec un faible nombre d'itérations donne une bonne estimation de la courbe de vraisemblance; cela prend son importance lorsqu'il s'agit de traiter les cas où il est difficile d'exécuter des millions d'itérations, à cause du nombre de marqueurs, ou du nombre de séquences. Nous avons montré comment simuler le temps auquel les événements se produisent, ceci afin de modéliser des tailles de population variables. Nous avons développé des équations s'appliquant au cas de croissance exponentielle, ce qui s'avère important en génétique, puisque la plupart des populations connaissent des croissances exponentielles.

L'algorithme de reconstruction des graphes fut ensuite présenté, et nous avons exploré diverses solutions pour diminuer les temps de calcul nécessaires à l'estimation. Nous avons obtenu de meilleurs résultats que ceux de l'algorithme "naïf", et nous avons montré comment diminuer encore le temps de calcul si le nombre de marqueurs n'est pas trop grand. Nous avons vu que sur des données réelles, le temps de calcul peut être problématique. Cependant, il serait sûrement possible d'améliorer encore la performance du programme en le faisant optimiser par un programmeur professionnel. Une solution complémentaire est de faire en parallèle les calculs sur différentes machines; il est facile et valide de combiner plusieurs simulations. Aussi, les laboratoires de recherche, académiques ou privés, susceptibles d'utiliser de telles méthodes ont des outils de calcul beaucoup plus puissants que ceux dont nous disposons pour ce travail. Enfin, en

cartographie génétique, les estimations sont beaucoup plus importantes que les temps de calcul: s'il existait une méthode "ultime" de cartographie génétique, peu de chercheurs se plaindraient des temps de calcul.

La méthode a ensuite été étendue à un autre type de marqueurs génétiques, les microsatellites, en développant un modèle de mutation pour ces marqueurs, puis en modifiant l'équation de récurrence. Ce nouveau modèle a ainsi pu être testé sur des ensembles de données réelles. On a vu que dans un cas la variation était trop grande pour pouvoir conclure quoi que ce soit, probablement dû à des données contenant trop peu d'informations pour notre méthode. Dans l'autre cas, celui de la fibrose kystique, l'analyse de l'échantillon avec seulement les cas possédant $\Delta F508$ a démontré que la méthode fonctionne bien puisque r_T est estimé très proche de sa valeur réelle. Les autres estimations, avec tous les cas, montrent toujours un pic de vraisemblance autour de $\Delta F508$, mais pas de façon consistante. Il est intéressant de noter que ces pics de vraisemblance sont présents même si 34% des cas ne possèdent pas la mutation $\Delta F508$, ce qui nous laisse espérer que la méthode pourrait être utilisée pour des maladies complexes.

Nous avons exploré diverses modifications, comme le changement des poids des recombinaisons, ou l'utilisation du nombre d'étapes maximum permises dans la reconstruction des graphes. De plus, nous avons proposé une modification consistant à ne tenir compte seulement que des parents potentiels pour déterminer les probabilités de recombinaison, et nous avons montré que cette procédure modifiée nous donnait de bien meilleurs résultats. Cette procédure est cependant mal justifiée théoriquement, et doit à ce titre être considérée comme exploratoire. Mais il est clair, d'après les résultats, que c'est une piste de recherche intéressante.

Enfin, nous avons développé l'idée du rééchantillonnage dans le graphe de recombinaison ancestral. Nous avons obtenu des résultats décevants pour le modèle à mutation non récurrente, mais des résultats intéressants pour le modèle avec microsatellite. Le rééchantillonnage est probablement une autre piste à explorer plus

en profondeur; plusieurs étapes de rééchantillonnage pourraient être effectuées, et le moment du rééchantillonnage dans la reconstruction du graphe pourrait être déterminé par le coefficient de variation des poids, et non fixé comme en ce moment. Évidemment, un plus grand nombre d'itérations devrait être utilisé pour faire des simulations.

Le graphe de recombinaison ancestral est un bon moyen de modéliser l'histoire de séquences apparentées. Comme nous l'avons vu, il est très facile d'étendre la méthode à différents types de marqueurs, pour diverses tailles de population variant dans le temps, ou pour toute autre situation comme par exemple, pour un autre processus de mutation ou pour tenir compte des "hot spots" de recombinaison. Malheureusement, beaucoup de variation est observée dans les courbes de vraisemblance. Nous pensons que modifier la distribution proposée qui construit les graphes pourrait grandement diminuer la variation. Cette distribution pourrait être la distribution de Fearnhead et Donnelly (2001), ou une distribution basée sur le travail exploratoire que nous avons fait en considérant les parents potentiels. Il est clair que si l'on arrive à éliminer ou diminuer la variation des courbes de vraisemblance, la méthode produirait des estimés ayant des intervalles de confiance très petits en comparaison avec d'autres méthodes. Par exemple, une des simulations faites par rééchantillonnage sur les données de la fibrose kystique avec $m = 10m$ et $n_S = 9$ (cf. fig. 5.8.11 c) donne une estimation de 0.879 Mb, située à seulement 0.001 kB de $\Delta F508$, avec un intervalle de confiance représentant à peine 0.40% de la longueur de la séquence totale, par rapport aux autres méthodes qui présentent des intervalles de confiance couvrant de 6% à 56% de la séquence. Pour un scientifique cherchant la mutation d'un gène qui cause une maladie, cette différence est énorme, en temps et en argent. Ainsi, notre méthode semble offrir une nouvelle information par rapport aux autres méthodes de cartographie. Notre recherche du TIM par intervalle est coûteuse en temps, mais précieuse en renseignements sur la position du TIM.

Bibliographie

- Baum, L.E. 1972. An inequality and associated maximization technique in statistical estimation for probalistic functions of Markov processes, *Inequalities* 3:1–8.
- Boehnke, M. 1994. Limits of resolution of genetic linkage studies: implications for the positional cloning of human diseases genes, *American Journal of Human Genetics*. 55:379–390.
- Briggs, K. M. 1998. W-ology, or, some exactly solvable growth models, <http://www.btexact.com/people/brigsk2/W-ology.html>.
- Casella, G., et Berger R.L. 1990. *Statistical inference*. Wadsworth & Brooks/Cole Advanced Books & Software, Pacific Grove, CA. 650 pp.
- Collins, A., et Morton, N. E. 1998. Mapping a disease locus by allelic association, *Proceedings of the National Academy of Sciences of the USA*. 95:1741–1745.
- Corless, R. M., Gonnet, G. H., Hare, D. E. G., Jeffrey, D. J., et Knuth, D. E. 1996. On the Lambert W Function, *Advances in Computational Mathematics*. 5:329–359.
- Chen, Y. et Liu, J. S. 2000. Discussion of the paper by Stephens and Donnelly, *Journal of the Royal Statistical Society, Series B*. 62: 644–645.
- Donnelly, P., et Tavaré, S. 1995. Coalescents and the genealogical structure under neutrality, *Annual Review of Genetics*. 29:401–421.
- Ethier, S. N., Griffiths, R. C. 1990. On the two-locus sampling distribution. *Journal Mathematical Biology*.29:131–159.
- Fallin, D., et Schork, N.J. 2000. Accuracy of Haplotype frequency estimation for biallelic loci, via the expectation-maximization algorithm for unphased Diploid Genotype Data, *American Journal of Human Genetics*. 67:947–959.
- Fearnhead, P., et Donnelly, P. 2001. Estimating recombination rates from population genetic data. *Genetics*. 159:1299–1318.
- Fu, Y.-X., et Li, W.-H. 1999. Coalescing into the 21st century: an overview and prospects of coalescent theory. *Theoretical Population Biology*. 56:1–10.
- Gabriel, S.B., Schaffner, S.F., Nguyen, H., Moore, J.M., Roy, J., Blumensteil, B., Higgins, J., DeFelice, M., Lochner, A., Faggart, M., Liu-Cordero, S.N.,

- Rotimi, C., Adeyemo, A., Cooper, R., Ward, R., Lander, E.S, Daly M.J., et Altshuler, D. 2002. The structure of haplotype blocks in the human genome, *Science*. 296(juin):2225–2229.
- Graham, J., et Thompson, E. A. 1998. Disequilibrium likelihoods for fine-scale mapping of a rare allele, *American Journal of Human Genetics*. 63:1517–1530.
- Griffiths, R. C. 1981. Neutral two-locus multiple allele models with recombination, *Theoretical Population Biology*. 19:169–186.
- Griffiths, R. C. 1989. Genealogical-tree probabilities in the infinitely-many-site model, *Journal of Mathematical Biology*. 27:667–680.
- Griffiths, R. C. 1991. The two-locus ancestral graph, in “Selected Proceedings of the Symposium on Applied Probability” (I. V. Basawa and R. L. Taylor, Eds.) , Sheffield, 1989, Institute of Mathematical Statistics. IMS Lecture Notes-Monograph.
- Griffiths, R. C. 2001. Ancestral inference from gene trees, in “Genes, Fossils, and Behaviour: An Integrated Approach to Human Evolution”. (P. Donnelly and R. Foley Eds.), NATO Science Series: Series A: Life Sciences, IOS Press.
- Griffiths, R. C., et Marjoram, P. 1996. Ancestral inference from samples of DNA sequences with recombination, *Journal of Computational Biology*. 3:479–502.
- Griffiths, R. C., et Marjoram, P. 1997. An ancestral recombination graph, in “IMA Volume on Mathematical Population Genetics”, (P. Donnelly and S. Tavaré, Eds.), pp. 257–270, Springer-Verlag, New-York.
- Griffiths, R. C., et Tavaré, S. 1994a. Ancestral inference in population genetics, *Statistical Science*. 9:307–319.
- Griffiths, R. C., et Tavaré, S. 1994b. Sampling theory for neutral alleles in a varying environment, *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*. 344:403-410.
- Griffiths, R. C., et Tavaré, S. 1994c. Simulating probability distributions in the coalescent, *Theoretical Population Biology*. 46:131–159.
- Griffiths, R. C., et Tavaré, S. 1996. Markov chain inference methods in population genetics, *Mathematical and Computer Modelling*. 23:141–158.
- Griffiths, R. C., et Tavaré, S. 1997. Computational methods for the coalescent, in “Progress in Population Genetics and Human Evolution” (P. Donnelly and S. Tavaré, Eds.), IMA Volumes in Mathematics and its Applications, Vol. 87, pp. 165–182, Springer-Verlag, Berlin.
- Griffiths, R. C., et Tavaré, S. 1998. The age of mutation in a general coalescent tree, *Stochastic Models*. 14:273–295.
- Hästbacka, J., de la Chapelle, A., Kaitila, I., Sistonen, P., Weaver, W., et Lander, E.S. 1992. Linkage disequilibrium mapping in isolated founder populations: diastrophic dysplasia in Finland. *Nature Genetics*. 2:204–211.

- Hästbacka, J., de la Chapelle, A., Mahtani, M.M., Clines, G., Reeve-Daly, M.P., Daly, M., Hamilton, B.A., Kusumi, K., Trivedi, B., Weaver, A., Coloma, A., Lovett, M., Buckler, A., Kaitila, I., et Lander E.S. 1994. The diastrophic dysplasia gene encodes a novel sulfate transporter: positional cloning by fine-structure linkage disequilibrium mapping. *Cell* 78:1073-1087.
- Hey, J., et Wakeley, J. 1997. A coalescent estimator of the population recombination rate, *Genetics*. 145:833–846.
- Hudson, R. R. 1983. Properties of a neutral allele model with intragenic recombination, *Theoretical Population Biology*. 23:183–201.
- Hudson, R. R. 1990. Gene genealogies and the coalescent process, in “Oxford Surveys in Evolutionary Biology” (D. Futuyma and J. Antonovics, Eds.), Vol. 7, pp. 1–44, Oxford Univ. Press, Oxford.
- Hudson, R. R. 2001a. Linkage disequilibrium and recombination. Chapter 11 in “Handbook of Statistical Genetics” (D. Balding, M. Bishop and C. Cannings Eds.), pp 309–324, Wiley, Chichester.
- Hudson, R. R. 2001b. Two-locus sampling distributions and their applications, *Genetics*.159:1805–1817.
- Hudson, R. R. 2002. Generating samples under a Wright-Fisher neutral model of genetic variation, *Bioinformatics*. 18:337–338.
- Hudson, R.R., et Kaplan, N.L. 1985. Statistical properties of the number of recombination events in the history of a sample of DNA sequences, *Genetics*. 111:147–164.
- Kerem B., Rommens, J.M., Buchanan, J.A., Markiewicz, D., Cox, T.K., Chakravarti, A. et Buchwald, M.1989. Identification of the cystic fibrosis gene: genetic analysis. *Science*.245: 1073-1080.
- Kingman, J. F. C. 1982. The coalescent, *Stochastic Processes and their Applications*. 13:235–248.
- Kuhner, M. K., Yamato, J., et Felsenstein, J. 2000. Maximum likelihood estimation of recombination rates from population data, *Genetics*. 156:1393–1401.
- Larribe, F. Lessard, S. et Schork, N. J. 2002. Gene mapping via the ancestral recombination graph, *Theoretical Population Biology*. 62:215–229.
- Lander, E. S., et Schork, N. J. 1994. Genetic dissection of complex traits, *Science*. 265(5181):2037–48.
- Lam, J. C., Roeder, K., et Devlin, B. 2000. Haplotype fine mapping by evolutionary trees, *American Journal of Human Genetics*. 66:659–673.
- Liu, J.S., Sabatti, C., Teng J., Keats, B. J. B., et Rish, N. 2001. Bayesian analysis of haplotypes for linkage disequilibrium mapping, *Genome Research*. 11:1716–1724.
- Liu, J. S. 2001. Sequential Monte Carlo in Action in “Monte Carlo Strategies in Scientific Computing” (Eds.), Chapitre 4, pp. 79–104, Springer-Verlag New-York Inc.

- Matsumoto, M., et Nishimura, T. 1998. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator, *ACM Transactions on Modeling and Computer Simulation*. 8(1):3–30.
- McPeck, M. S., et Strahs, A. 1999. Assessment of linkage disequilibrium by the decay of haplotype sharing, with application to fine-scale genetic mapping, *American Journal of Human Genetics*. 65:858–875.
- Mendel, G. 1866. Experiments in plant hybridisation, *J. H. Bennet ed., 'English translation and commentary by R. A Fisher'*, Olivier and Boyd, Edinburg, 1965.
- Moran, P. 1962. The statistical processes of evolutionary theory, *Clarendon Press*, Oxford.
- Morris, A. P., Whittaker, J. C., et Balding, D. J. 2000. Bayesian fine-scale mapping of disease loci by hidden Markov models, *American Journal of Human Genetics*. 67:155–169.
- Morris, A. P., Whittaker, J. C., et Balding, D. J. 2002. Fine-scale mapping of disease loci via shattered coalescent modeling of genealogies, *American Journal of Human Genetics*. 70:686–707.
- NBIF (National Biotechnology Information Facility) / Regents of the New Mexico State University, 2001. *EXTented NUMeric (EXTNUM) range floating point type*. <http://www.nbif.org/products/bioinfo/extnum/extnum.php>.
- Neuhauser, C. 2001. Mathematical models in population genetics. Chapter 6 in “Handbook of Statistical Genetics” (D. Balding, M. Bishop and C. Cannings Eds.), pp 153–177, Wiley, Chichester.
- Nielsen, R. 2000. Estimation of population parameters and recombination rates from single nucleotide polymorphisms, *Genetics*. 154:931–942.
- Nordborg, M. 2001. Coalescent theory. Chapter 7 in “Handbook of Statistical Genetics” (D. Balding, M. Bishop and C. Cannings Eds.), pp 179–212, Wiley, Chichester.
- Nordborg, M., et Tavaré, S. 2002. Linkage disequilibrium: what history has to tell us, *Trends in Genetics*. 18:83–90.
- Ott, J. 1999. Analysis of human genetic linkage. *The John Hopkins University Press*.
- Pennacchio, L. A., Lehesjoki, A. E., Stone, N. E., Willour, V. L., Virtaneva, K., Miao, J., et D’Amato, E. 1996. Mutations in the gene encoding cystatin B in progressive myoclonus epilepsy (EPM1), *Science*. 271:1731–1734.
- Pritchard, J. K., Seielstad, M. T., Perez-Lezaun, A. et Feldman, M. W. 1999. Population growth of human Y chromosomes: a study of Y chromosome microsatellites, *Molecular Biology and Evolution*. 6:1791–1798.
- Rannala, B., et Reeve, J.P. 2001. High resolution multipoint linkage disequilibrium in the context of a human genome sequence, *American Journal of Human Genetics*. 69:159–178.

- Rannala, B., et Slatkin, M. 1998. Likelihood analysis of disequilibrium mapping, and related problems, *American Journal of Human Genetics*. 62:459–473.
- Service, S. K., Temple Lang, D. W., Freimer, N. B., et Sandkuijl, L. A. 1999. Linkage-disequilibrium mapping of disease genes by reconstruction of ancestral haplotypes in founder population, *American Journal of Human Genetics*. 64:1728–1738.
- Schork, N. J., et Chakravarti, A. 1996. A nonmathematical overview of modern gene mapping techniques applied to human diseases. Chapter 2, in *Molecular genetics and gene therapy of cardiovascular disease*. Marcel Dekker Inc. New-York.
- Slatkin, M., et Rannala, B. 1997. Estimating the age of alleles by use of intraallelic variability, *American Journal of Human Genetics*. 60:447–458.
- Slatkin, M., et Hudson, R. R. 1991. Pairwise Comparisons of Mitochondrial DNA Sequences in Stable and Exponentially Growing Populations, *Genetics*. 129:555–562.
- Stephens, M., et Donnelly, P. 2000. Inference in molecular population genetics, *Journal of the Royal Statistical Society. Series B*. 62:605–655.
- Stephens, M., Smith, N.J., et Donnelly, P. 2001. A new statistical method for haplotype reconstruction, *American Journal of Human Genetics*. 68:978–989.
- Stephens, M. 2001. Inference under the coalescent. Chapter 8 in “Handbook of Statistical Genetics” (D. Balding, M. Bishop and C. Cannings Eds.), pp 213–238, Wiley, Chichester.
- Terwilliger, J. D. 1995. A powerful likelihood method for the analysis of linkage disequilibrium between trait loci and one or more polymorphic marker loci, *American Journal of Human Genetics*. 56:777–787.
- The cytogenetics gallery, Département de pathologie, Université de Washington: “<http://www.pathology.washington.edu/galleries/Cytogallery/cytogallery.html>”.
- Virtaneva, K., Miao, J., Träskelin, A.-L., Stone, N., Warrington, J. A., Weissenback, J., Myers, R. M., Cox, D. R., Sistonen, P., de la Chapelle, A., et Lehesjoki, A.-E. 1996. Progressive myoclonus epilepsy EMP1 locus to a 175kb interval in distal 21q, *American Journal of Human Genetics*. 58:1247–1253.
- Wagner, R. 2001. Mersenne twister random number generator, <http://www-personal.engin.umich.edu/~wagnerr/MersenneTwister.html>.
- Wall, J. D. 2000. A comparaison of estimators of the population recombination rate, *Molecular Biology and Evolution*. 17:156–163.
- Wall, J. D. 2003. Estimating ancestral population sizes and divergence times, *Genetics*. 163:395–404.
- Weissten, E. 2000. Wolfram Research, <http://math-world.wolfram.com/LambertsWFunction.html>.

-
- Xiong, M., et Guo, S. W. 1997. Fine-scale genetic mapping based on linkage disequilibrium: theory and applications, *American Journal of Human Genetics*. 60:1513–1531.

Annexes

Annexe A

Données de l'exemple E

Nous présentons dans cette annexe les données des séquences des cas de fibrose kystique. Les quatre matrices qui suivent contiennent en ligne une séquence. Pour chaque ligne, on trouve de gauche à droite, le numéro de la séquence i , la multiplicité de la séquence, le statut cas/témoin (nous ne présentons que les cas puisque nous n'utilisons qu'eux), et la suite des 23 marqueurs observés. Ces données proviennent de Kerem *et al* (1989).

i	n_i	<i>Statut</i>	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}
1	1	1	0	0	0	0	0	0	0	1	1	1	1
2	1	1	0	0	0	0	0	0	0	—	1	1	1
3	1	1	0	0	0	1	1	1	1	0	0	0	0
4	1	1	0	0	1	0	0	0	0	0	0	0	0
5	1	1	0	0	1	0	0	0	0	0	0	0	0
6	3	1	0	0	1	0	0	0	0	0	1	1	1
7	1	1	0	0	1	0	0	0	0	1	0	0	0
8	1	1	0	0	1	0	0	0	0	1	0	0	0
9	1	1	0	0	1	0	0	0	0	1	1	1	1
10	1	1	0	0	1	0	0	1	1	1	1	1	1
11	1	1	0	0	1	0	0	1	1	—	0	0	0
12	1	1	0	0	1	0	0	1	1	—	0	0	0
13	2	1	0	0	1	1	1	0	0	0	1	1	1
14	1	1	0	0	1	1	1	0	0	0	1	1	1
15	1	1	0	0	1	1	1	0	0	—	1	1	1
16	1	1	0	1	0	0	0	0	0	1	1	1	1
17	1	1	0	1	0	0	0	0	0	—	1	1	1
18	1	1	1	0	0	0	0	0	0	0	0	0	0
19	1	1	1	0	0	0	0	0	0	0	1	1	1
20	1	1	1	0	0	0	0	0	0	1	0	0	0
21	1	1	1	0	0	0	0	0	0	1	1	1	0
22	1	1	1	0	0	0	0	0	0	—	0	0	0
23	1	1	1	0	0	0	0	0	0	—	1	1	1
24	1	1	1	0	0	0	0	0	0	—	1	1	1
25	1	1	1	0	0	0	0	1	1	0	0	0	0
26	1	1	1	0	0	1	1	0	0	1	0	0	0
27	1	1	1	0	1	0	0	0	0	0	0	0	0
28	1	1	—	—	—	—	—	—	—	1	0	0	0
29	2	1	0	0	1	0	0	0	0	0	1	1	1
30	1	1	0	0	1	0	0	0	0	1	1	1	1

i	n_i	<i>Statut</i>	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}
31	1	1	0	0	1	0	0	0	0	1	1	1	1
32	4	1	0	0	1	0	0	1	1	0	1	1	1
33	1	1	0	0	1	0	0	1	1	0	1	1	1
34	1	1	0	0	1	0	0	1	1	0	1	1	1
35	1	1	0	0	1	0	0	1	1	0	1	1	1
36	1	1	0	0	1	0	0	1	1	—	1	1	1
37	1	1	0	0	1	0	0	1	1	—	1	1	1
38	1	1	0	0	1	1	1	0	0	0	1	1	1
39	1	1	0	0	1	1	1	0	0	—	1	1	1
40	1	1	0	1	0	0	0	0	0	0	1	1	1
41	1	1	0	1	0	1	1	0	0	0	1	1	1
42	1	1	0	1	0	1	1	0	0	1	1	1	1
43	1	1	1	0	0	0	0	0	0	0	1	1	1
44	1	1	1	0	0	0	0	0	0	—	0	0	1
45	1	1	1	0	0	0	0	0	1	0	1	1	1
46	10	1	1	0	0	0	0	1	1	0	1	1	1
47	11	1	1	0	0	0	0	1	1	0	1	1	1
48	3	1	1	0	0	0	0	1	1	0	1	1	1
49	1	1	1	0	0	0	0	1	1	0	1	1	1
50	1	1	1	0	0	0	0	1	1	0	1	1	1
51	4	1	1	0	0	0	0	1	1	—	1	1	1
52	1	1	1	0	0	0	0	1	1	—	1	1	1
53	1	1	1	0	0	1	0	1	0	1	1	1	1
54	2	1	1	0	0	1	1	0	0	0	1	1	1
55	1	1	1	0	0	1	1	0	0	0	1	1	1
56	2	1	1	0	0	1	1	0	0	—	1	1	1
57	1	1	1	0	0	1	1	1	1	0	1	1	1
58	1	1	1	0	0	1	1	1	1	0	1	1	1
59	1	1	1	0	0	1	1	1	1	0	1	1	1
60	1	1	1	0	1	0	0	1	1	—	1	1	1
61	1	1	1	1	—	1	1	0	0	0	1	1	1

i	s_{12}	s_{13}	s_{14}	s_{15}	s_{16}	s_{17}	s_{18}	s_{19}	s_{20}	s_{21}	s_{22}	s_{23}
1	1	1	1	1	0	0	1	0	0	1	1	1
2	1	1	1	1	1	1	0	1	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	1	0
4	1	0	1	1	0	0	0	0	0	1	1	1
5	1	0	1	1	1	1	0	1	1	0	0	0
6	1	1	1	1	1	1	0	1	0	0	0	0
7	0	0	0	1	0	0	0	0	0	1	1	1
8	—	—	—	0	0	0	0	0	0	0	1	0
9	1	1	1	1	0	0	1	1	0	1	1	1
10	1	1	1	1	0	0	1	1	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0
12	—	—	—	0	0	0	0	—	—	0	0	0
13	1	1	1	1	0	0	1	1	0	0	0	0
14	1	1	1	1	0	0	1	1	0	1	1	1
15	1	1	1	1	0	0	1	1	0	0	0	0
16	1	1	1	1	0	0	1	1	0	1	1	1
17	1	1	1	1	1	1	0	1	0	0	0	0
18	0	0	0	0	0	0	0	0	1	0	1	1
19	1	1	1	1	1	1	0	1	0	—	—	—
20	—	—	—	—	—	—	0	—	—	1	1	1
21	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	1	0	0	0
23	1	1	1	1	0	0	1	1	0	0	0	0
24	1	1	1	1	1	1	0	1	0	1	1	1
25	0	0	0	1	0	0	1	1	0	—	—	—
26	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	1	1	1
28	0	0	0	0	0	0	0	0	0	0	0	0
29	1	1	1	1	1	1	0	1	0	1	1	1
30	1	1	1	1	0	0	1	1	0	0	0	0

i	s_{12}	s_{13}	s_{14}	s_{15}	s_{16}	s_{17}	s_{18}	s_{19}	s_{20}	s_{21}	s_{22}	s_{23}
31	—	—	—	1	1	1	0	—	—	0	0	0
32	1	1	1	1	1	1	0	1	0	1	1	1
33	1	1	1	1	1	1	0	1	0	—	—	—
34	—	—	—	1	1	1	0	1	0	1	1	1
35	—	—	—	1	1	1	0	—	—	1	1	1
36	1	1	1	1	1	1	0	1	0	0	0	0
37	1	1	1	1	1	1	0	1	0	1	1	1
38	1	1	1	1	1	1	0	1	0	1	1	1
39	1	1	1	1	1	1	—	1	0	0	0	0
40	1	1	1	1	1	1	0	1	0	0	0	0
41	—	—	—	1	1	1	—	1	0	1	1	1
42	1	1	1	1	1	1	0	1	0	1	1	1
43	1	1	1	1	1	1	0	1	0	0	0	0
44	—	—	—	1	1	1	0	1	0	0	0	0
45	1	1	1	1	1	1	0	1	0	1	1	1
46	1	1	1	1	1	1	0	1	0	0	0	0
47	1	1	1	1	1	1	0	1	0	1	1	1
48	1	1	1	1	1	1	—	1	0	1	1	1
49	—	—	—	1	1	1	0	1	0	0	0	0
50	—	—	—	1	1	1	0	—	—	1	1	1
51	1	1	1	1	1	1	0	1	0	0	0	0
52	1	1	1	1	1	1	0	1	0	0	1	0
53	1	1	1	1	1	1	0	1	0	0	1	1
54	1	1	1	1	1	1	0	1	0	0	1	1
55	1	1	1	1	1	1	0	1	0	1	1	1
56	1	1	1	1	1	1	0	1	0	0	1	1
57	1	1	1	1	1	1	0	1	0	0	0	0
58	1	1	1	1	1	1	0	1	0	1	1	1
59	—	—	—	1	1	1	0	1	0	0	0	0
60	1	1	1	1	1	1	0	1	0	0	0	0
61	—	—	—	—	—	—	0	—	—	1	1	1

Annexe B

Manuel d'instruction du programme

Gene Mapping via The Ancestral Recombination Graph

Short manual - version 0.91

Program version 52.03 for SNPs (**MapArgS**),
m123 for microsatellites (**MapArgM**)

Saturday, August 9, 2003

FABRICE LARRIBE

UNIVERSITÉ DE MONTRÉAL

larribe@dms.umontreal.ca

This **short** manual explains how to install and how to use the program **MapArg** (Mapping with the Ancestral Recombination Graph). Two versions are available: **MapArgS** for use with SNPs markers, and **MapArgM** for use with microsatellite markers. After explaining how to install and run the program, what the input should be, and how to exploits results, we will process an example from the beginning to the end. The data and parameter file of the example are in the package, and the first thing to do would be to try it, to make sure that the program is working properly on your machine.

0. Installation	3
1. Data file	5
2. Parameter file	5
3. Command line	7
4. Output Files	8
5. MapArgS example	9
6. Convergence aspects	12
7. MapArgM example	13
8. Notes	14
8.1. Calculus of κ (kappa) for the variable population size model	14
8.2. The μ (mu) parameter for SNPs	15
8.3. The ReWeight (ξ) parameter	15
8.4. The MaxIter parameter	15
8.5. How to combine results from different runs ?	16
8.6. Trace of estimation	16
8.5. Program errors and remarks	17
9. T _E X output example	18

0. Installation

The programs are written in C++, and compile correctly with the GNU version of C++. If you use Unix or Linux, a C++ compiler should already be installed. If you are within a Windows environment, you can obtain the GNU distribution by installing Cygwin, which is a free “unix like” environment for Windows (“<http://www.cygwin.com/>”). Of course, others C++ compilers can be used, but some additional work will probably be needed in order the programs to compile correctly. The installation procedure of the programs is the is the same for both cases, Windows or Unix (Linux).

Desktop computers usually have a very limited range of admissible numbers that they can manipulate. As we work with likelihoods of very small value, the program can use a library (the “extnum” library) to extend the possible numbers in order that the likelihood may be evaluated correctly. In fact, if the data consists of only a few sequences and a few markers (say 30 sequences and 5 markers), you do not need to use the extnum library. When used, the program can manipulate (positive or negative) numbers between 1×10^{-646456993} and 2×10^{646456992} , which is enough for our needs. The drawback is that the program is a little slower to run. So, before beginning installation, you should download the extnum library from “<http://www.swbic.org/products/bioinfo/extnum/extnum.php>” after having accept the license agreement. In the archive, you should have all these files:

- 1) the main C++ programs: `52_03.cpp` et `micro123.cpp`
- 2) the secondary C++ programs: `combine16.cpp` et `combine-m02.cpp`
- 3) `example.par`, `example.par`, `peter.par` and `peter.dat` files
- 4) `manual.pdf`
- 5) Mersenne Twister header file, `MersenneTwister.h`
- 6) A matrix library, `matrix.h`

The secondary C++ programs, `combine16.cpp` et `combine-m02.cpp`, are used to combine results from different simulations, we will show how to use them later. Let’s now see how to compile the programs. If the extnum library

is used, remove the comment (`//`) at the beginning of the program file at the line `#define Extnum 1`. If the `extnum` package is used, you should first find the right version for you (there are two versions), depending on the type of machine you use. To do that, follow these steps:

- 1) `g++ -c real.cc`
- 2) `g++ -c DblBits_LittleEndian.cc`
- 3) `g++ -c DblBits_BigEndian.cc`
- 4) `g++ test.cc real.o DblBits_LittleEndian.o -o test_Little.exe`
- 5) `g++ test.cc real.o DblBits_BigEndian.o -o test_Big.exe`
- 6) `./test_Little.exe`
- 7) `./test_Big.exe`

One of two tests (previous item 6 or 7) should show you a list of operations where tests are PASSED, while the other deliver WARNINGS. You must use, of course, the version which gives you the appropriate PASSED results.

- 8) Now, if you use the `LittleEndian` version, run the following commands:


```
g++ 52_03.cpp real.o DblBits_LittleEndian.o -O3 -o MapArgS.exe
g++ m123.cpp real.o DblBits_LittleEndian.o -O3 -o MapArgM.exe
g++ combine16.cpp real.o DblBits_LittleEndian.o -O3 -o CombS.exe
g++ combinem-02.cpp real.o DblBits_LittleEndian.o -O3 -o CombM.exe
```

 or, if you use the `BigEndian` version, run the following commands:


```
g++ 52_03.cpp real.o DblBits_BigEndian.o -O3 -o MapArgSNPs.exe
g++ m123.cpp real.o DblBits_BigEndian.o -O3 -o MapArgM.exe
g++ combine16.cpp real.o DblBits_BigEndian.o -O3 -o CombS.exe
g++ combinem-02.cpp real.o DblBits_BigEndian.o -O3 -o CombM.exe
```
- 9) You are ready to use `MapArgS.exe` and `MapArgM.exe`.

Notes on others tools that are used in the program:

- 1) Mersenne Twister, a random number generator. Information on the implementation on the Mersenne twister can be found at: "<http://www-personal.engin.umich.edu/~wagnerr/MersenneTwister.html>". Licence GNU: see the licence in the `MersenneTwister.h` file.
- 2) Extnum Library Information on the Extnum Library can be found at: <http://www.swbic.org/products/bioinfo/extnum/extnum.php>. You should accept licence agreement before you use it (Thanks are extended to the *Southwest Biotechnology and Informatics Center*).
- 3) A simple matrix library for the microsatellite version of the program: informations can be found at "<http://www.techsoftpl.com/matrix/>". This library is free for educational and non-commercial use.

1. Data file

This is an ascii (text) file. Each line is a sequence. Each value is separated from another value by one or more spaces. The first value on each line is the multiplicity of the sequence. The second value is the trait: 1 for case, 0 for control. The others values are then the marker loci. With **MapArgS**, the program recognizes only binary markers, like SNPs. A mutation (non-ancestral) should be code 1, an ancestral marker should be coded 0. With **MapArgM**, the values of the markers loci are integer, and any values between 0 and 10 can be used. Use the code “-” for missing value for both programs.

To code the sequences for **MapArgS**, if you have cases and controls, you can use the most frequent allele value in the control population to determine the ancestral allele. If only cases are present, use the most frequent allele in the cases.

2. Parameter file


The parameter file contains all the parameters of the process. This is a an ascii (text) file. Each line must begin with a parameter name. Here is a list of admissible parameters; at the end of each line is the type of the parameter: [int] for integer, [real] for real, [char] for text, **vector of [X]** for a vector a variables of type X. The line should finish exactly after the parameter value: if a space is added after a value, the program could crash. If the # symbol is added at a beginning of a line in the parameter file, this line is ignored: this permits to put some comments in the file. [⊛] means that the option ought to be handled with caution]

Ne Effective Population Size (Default ≈ 10000) [int]

mu Mutation rate in one sequence. Used to calculate the parameter θ , such that

$$\theta = 4N_e\mu \text{ [real] (see 6.3)}$$

Dist Distance in Morgans between marker loci. There should be exactly $L - 1$ distances, if there are L marker loci. **vector of [real]**

- NbSim** The number of iterations of the process. [int]
- NbCan** The number of total points along the sequence of which you want to evaluate the likelihood. The number of candidates by interval is then proportional to the distance for each interval. [int]
- VNbCan** A vector of the number of candidates for each interval. If this line is empty, this vector is calculated from the **NbCan** parameter. There should be exactly $L - 1$ values, if there are L marker loci. A value could be 0 if you do not wish to evaluate likelihood in a certain interval. vector of [int]
- ReWeight** An option to modify the proposal distribution to construct the graph. If the value is 0.5 for example, the probabilities that a recombination happens at each step of the graph is divided by two, and the functional value of the likelihood is adjusted. This option should be set with care: it is an explanatory option. Default value is one (1). $0 < [\text{real}] \leq 1$  See (6.4)
- ReType** Admissible values are 1 or 0. A value of 0 means that the number of recombination events at each step is the number of sequences times the number of possible intervals for recombination (if all intervals are ancestral). A value of 1 means that the number of recombination events is the number of intervals: the probability of recombination is calculated for each interval (taking into account all information), and if a recombination event is chosen by the process, a sequence is chosen randomly to recombine. With this last option, estimations are faster, but we seem to have more variability. Default is 0. [0/1]
- kappa** The parameter of the exponential for a variable population size. If the value is 0, a constant population size is assumed. [real] (See 6.1)
- ResultFile** The output file for the results. [char]
- DataFile** The name of the file containing the data. [char]
- PrintNbSim** To have an idea of the current speed of execution, the program outputs the number of the current graph reconstructed during process **PrintNbSim** times by interval (see example later in this manual). Default

is 10. [int]

MaxIter The maximum number of iterations to construct a graph. [int] ☆ (See 6.5)

Phase Resampling status: -1 represents no sampling/resampling, 0 is for sampling, 1 is for resampling. [-1/0/1]

N The sample size used for resampling [int]

3. Command line

The program is run from a dos box in Windows, or from a shell in Unix. Some options could be set in the parameter file, or by command line arguments, but the last ones have always priority. Options are:

- p** Parameter file.
- y** Data file.
- K** Number of simulations. This number could be, for example, 5, 10, 1m (for 1 thousand), 56m, 1M (for 1 million) or 1.5M.
- o** Output file.
- e** Request for calculation of the expected time to do a certain number of iterations. (This option does not work on all systems. Number should be in the same format as the **-K** option.)
- r** Seed of the random number generator. If no seed is given, the seed is based on time. If you set a seed, several runs (with the same parameters) will give the same result.
- P** Resampling: -1 is no sampling or resampling, 0 is sampling, 1 is resampling.
- N** Size of the sample to resample (must be used in sampling and resampling)
- G** Resampling: number of graphs to retain.
- Z** Resampling: file which contains information on the sampled graphs (must be use in sampling and resampling).

The minimum command line to run the program is of the form:

```
MapArgS.exe -p example.par
```

Some more elaborated command lines are for example:

```
MapArgS.exe -p example.par -K 10m -e 1M -o ResExample -r 1234
```

meaning that the parameter file is `example.par`, you want to calculate the expected time to do 1×10^6 iterations based on ten thousand iterations, you want that the output file to be named `ResExample.txt`, and initiate the process with the seed 1324. Another example is:

```
MapArgS.exe -p example.par -K 2M -P 0 -N 5 -Z ResSample
```

meaning that you want to run 2 millions iterations, to do sampling at size $n = 5$, and the name the file that keeps information used for resampling to be `ResSample`. Now, do to resampling, issue the command:

```
MapArgS.exe -p example.par -K 2M -P 1 -N 5 -Z ResSample -o ResX
```

4. Output Files

The output file contains many informations about the data, parameters and results. Four files are actually produced automatically. If the output file is `ResEx` for example, then the four files `ResEx.txt`, `ResEx.plt`, `ResEx.don` and `ResEx.tex` are produced. The `ResEx.txt` file contains information about parameters, data, and the estimated likelihood curve, with the maximum likelihood estimator of r_T . The `ResEx.don` file contains only the data which is necessary to construct the likelihood curve; this data could be imported directly to another software to draw graphs, using space as separators. The `ResEx.plt` is a small command file which will plot the likelihood and the relative likelihood curves using `GnuPlot`. This software is a free utility to draw graphs, and is available for many operating systems, like Windows, Unix, Macintosh and BeOs (<http://www.gnuplot.info/>), so if you system is configured properly,

a simple click on the `ResEx.plt` file will open a window with the representation of the log-likelihood curve. Finally, the `ResEx.tex` file is a pre-formatted \TeX file which shows parameters, estimate, descriptive statistics about graphs reconstruction, and the plot of the log-likelihood curve. The `Dra \TeX` package, which is freely available from the Comprehensive \TeX Archive Network (CTAN) (<http://www.tug.org/ctan.html>), is required to compile the file. This file is the most informative output and can be printed easily. An example of such a file is presented at the end of the manual. Some options are available at the beginning of the file to obtain only the log-likelihood profile in landscape format, to add or remove vertical lines between intervals, to draw confidence intervals, etc. The code could be copied and pasted into a \TeX document to insert the figure.

5. MapArgS example

In this example, we have $n = 10$, 6 different sequences types, 8 cases and 2 controls. We have four marker loci, so we will look for the TIM in each of the three intervals.

- The data file, named `peter.dat`, appears like this:

```
5 1 0 0 0 0
1 0 0 0 0 1
1 1 0 0 0 1
1 1 0 0 1 0
1 1 1 0 0 0
1 0 1 1 0 1
```

- The parameter file, named `peter.par`, appears like this:

```
mu 0.000125
Ne 10000
Dist .000007382 .000096382 .000021236
NbSim 100000000
NbCan 100
#VNbCan 0 27 0
ReWeight 1
```



```

kappa 0
ResultFile peter.txt
DataFile peter.dat
MaxIter 500
Phase -1
ReType 0
TeXOutput 1

```

- The command line is:

```
MapArgS.exe -p peter.par -o ResSp1 -K 5m -r 1 -e 1M
```

- During execution, the program outputs to the screen, as it runs:

```

Interval 1 [5m] Sim 500 1m 1.5m 2m 2.5m 3m 3.5m 4m 4.5m 5m
Interval 2 [5m] Sim 500 1m 1.5m 2m 2.5m 3m 3.5m 4m 4.5m 5m
Interval 3 [5m] Sim 500 1m 1.5m 2m 2.5m 3m 3.5m 4m 4.5m 5m
[CPU 0j 0h 0m 11s. / Real time: 0j 0h 0m 12s.]
[ Thu Aug 7 00:35:40 2003 / Thu Aug 7 00:35:52 2003 ]
+ Total estimated time for 1M simulations is 38 m 17 s.

```

And so, after about 12 seconds, the program ends: the output files are created. As we have requested an estimate of the time required to do 1 million iterations is output, we can see that 38 minutes would be necessary.

- The file `ResSp1.txt` look like this:

```

HapScan [./MapArgS] (v. 52.03)
+ Data File:peter.dat
+-----+
NT = 6 / NM = 4 / L = 5 / Sample Size: 10
Sequence ... 0 1 2 3 4 5
Multiplicite 5 1 1 1 1 1
Trait ..... 1 0 1 1 1 0
M1 ..... 0 0 0 0 1 1
M2 ..... 0 0 0 0 0 1
M3 ..... 0 0 0 1 0 0
M4 ..... 0 1 1 0 0 1
+ Parameter File:peter.par
+-----+
Sample Size:10 for 6 different sequences
r=0.000125 / rho=5 / theta=5 / mu=0.000125 / mu_l=2.5e-05
kappa=0
Likelihood on 5000 simulations. ReWeight = 1
[MaxIter 500 ]
RandomSeed: 1
Model for recombination: Original.
[Min(L)=2.17835e-12 / Max(L)=5.78793e-09]
Candidates [ 7 77 17 ]
Driving Value [ 0.0003691 0.0055573 0.0114382 ]
DropOut [ 0 0 0 ]

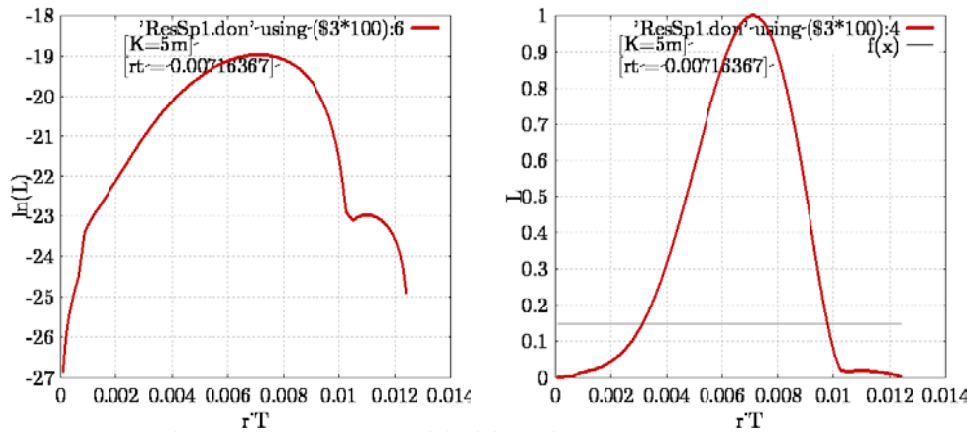
```

```

CPU Time process: 0j 0h 0m 11s.
Human Time process: 0j 0h 0m 12s.
+ Results:
+-----+
Can Int  Coordinate  Lik_R          Lik           Ln(L)
  1  1    9.2275e-07    0.00037636    2.17835e-12   -26.8525
  2  1    1.8455e-06    0.000812671   4.70368e-12   -26.0827
  3  1    2.76825e-06    0.0013089    7.57584e-12   -25.6061
  4  1     3.691e-06    0.00186503    1.07946e-11   -25.252
  5  1    4.61375e-06    0.00248101    1.43599e-11   -24.9666
  6  1    5.5365e-06    0.00315683    1.82715e-11   -24.7257
  7  1    6.45925e-06    0.00389245    2.25292e-11   -24.5162
  8  2    8.61767e-06    0.0118379    6.85172e-11   -23.4039
  9  2    9.85333e-06    0.0141499    8.18986e-11   -23.2255
 10  2    1.1089e-05     0.0165074    9.55434e-11   -23.0714
.../...  .../...  .../...  .../...  .../...
 82  2    0.000100057    0.0710581    4.11279e-10   -21.6117
 83  2    0.000101293    0.0402191    2.32785e-10   -22.1809
 84  2    0.000102528    0.0193229    1.1184e-10    -22.914
 85  3    0.000104944    0.0158356    9.16553e-11   -23.113
 86  3    0.000106124    0.0168592    9.75797e-11   -23.0504
 87  3    0.000107303    0.0175754    1.01725e-10   -23.0087
.../...  .../...  .../...  .../...  .../...
 98  3    0.000120281    0.00928424   5.37365e-11   -23.6469
 99  3    0.000121461    0.00727125   4.20855e-11   -23.8913
100  3    0.00012264    0.00505618   2.92648e-11   -24.2546
101  3    0.00012382    0.00263453   1.52485e-11   -24.9065
+ Maximum Likelihood Estimate:
+-----+
Candidate 59 in interval 2 at position 0.00716367cM
+ Descriptive Statistics.
+-----+
ESS [ 6.89007 1.26095 2.83733 ]
Nb Graph really done [ 5000 5000 5000 ]
Coalescence Event: Mean [ 41.0152 18.6888 24.8924 ]
                   Std  [ 23.6274 3.99319 9.03688 ]
Recombinat. Event: Mean [ 32.0152 9.6888 15.8924 ]
                   Std  [ 23.6274 3.99319 9.03688 ]
Mutation Event: Mean [ 5 5 5 ]
                   Std  [ 0 0 0 ]
Number Events by graph: [ 78.0304 33.3776 45.7848 ]
                   Std  [ 47.2548 7.98638 18.0738 ]

```

- The log-likelihood (left) and the relative log-likelihood (right) curves obtained by GnuPlot look like this:



We can see that the maximum likelihood estimate is $\hat{r}_T \approx 0.007$.

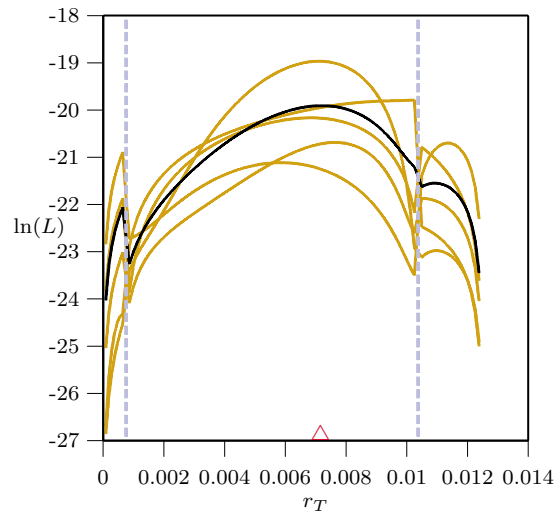
6. Convergence aspects

A major issue is how many simulations have to be made in order to achieve convergence. If it is possible, several millions of simulations are preferable, but in some cases, only a few thousands can be done in a reasonable time. Currents results show that some non-negligible variability is present in the estimation process, whatever the number of simulations, but this variability tends to decrease slowly by increasing the number of iterations. One simple thing that could be done is to run several processes and compare the likelihood profiles; if they are similar, then convergence is probably achieved. For example, with the supplied data, we can run these commands:

```
MapArgS.exe -p peter.par -o ResSp1 -K 5m -r 1
MapArgS.exe -p peter.par -o ResSp2 -K 5m -r 2
MapArgS.exe -p peter.par -o ResSp3 -K 5m -r 3
MapArgS.exe -p peter.par -o ResSp4 -K 5m -r 4
MapArgS.exe -p peter.par -o ResSp5 -K 5m -r 5
combineS.exe -i ResSp1 ResSp2 ResSp3 ResSp4 ResSp5 -o ReSp12345
-A
```

Here we are estimating the same likelihood five times, with different seeds. You do not need to choose a seed; we have just chosen some seeds here so that you can replicate the results in your installation. After the five estimations, the `combineS` program is used to combine the different results. Syntax is simple: `-i` for the input files, `-o` for the output file, and the option `-A` to draw all the curves

(-0 would have produced only the resulting curve). The plot of the `tex` file is shown in the figure that follows:



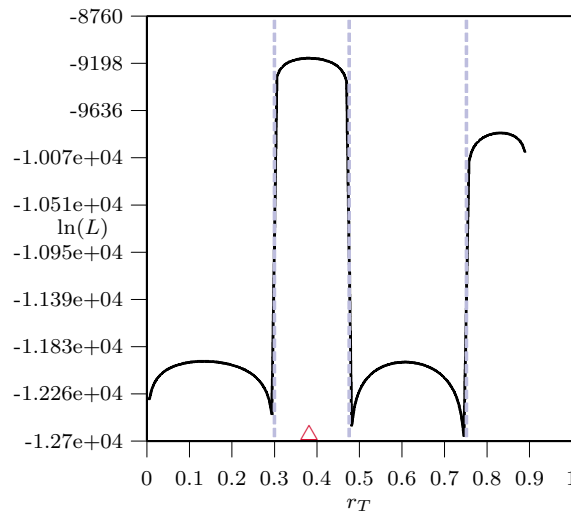
As we can see, the five likelihood profiles are quite similar, but variability is present. In fact, millions of iterations give almost the same results, but with less variability; this does not mean that 25m iterations are enough, and in general, this is not the case. A similar program, named `CombineM.exe` can be used in combination with the the microsatellite program.

7. MapArgM example

The `MapArgM` program is basically an adaptation of the `MapArgS` program, so, all parameters are almost the same. We will just highlight the differences, and present a small example, so you can check if your installation is OK. Two new parameters are introduced. The `MuType` parameter is the equivalent of the `ReType` parameter, but for mutations, instead of recombinations. A value of 0 means that we evaluate all possible mutations events at each step, and a value of 1 means we only evaluate the possible mutation events by sequence, and if a mutation event is chosen, then a marker to mutate is chosen. `MuType=1` is a simplification, and the program is far faster. For many sequences and markers, to evaluate all the possibles mutations events is difficult. The other different parameter from `MapArgS` is `MuOrder`. This is the biggest step in the change in the repeat number of a microsatellite marker in a mutation event; so, if for

example `MuOrder=2` and the current state of an allele of a given sequence is 4, a mutation is possible from that allele to the values 2,3,5 and 6. If `MuOrder=1`, then mutations are only possible to the values 3 and 5.

An example of a microsatellite data, and its parameter file are included in the package; these files are `exampleM.dat` and `exM.dat`. If you run the command: `MapArgM.exe -p exampleM.par -K 100 -o resM -e 1M -r 1234`, the program ends after about four minutes, and you have an estimate of the likelihood curve. Do note that on my system, the program estimates that the time required to execute 1M of simulations is about 28 days. If you edit the file `resM.txt`, you can see that the maximum likelihood estimate is in interval 2, at the coordinate 0.382133cM. The next figure shows the likelihood profile.



8. Notes

8.1 Calculus of κ for the variable population size model.

We illustrate the calculus of κ (**kappa**) using the example of the finnish population. According to Virtaneva *et al.* (1996), the finnish population was founded 2000 years ago by 1000 individuals, and now, the population of Finland is around 5×10^6 . Following the notation in Larribe *et al.* (2002), we have:

$$N(t) = N(0) \exp(-\kappa t).$$

So, at $t = 2000$, we have;

$$1000 = 5 \times 10^6 \exp(-\kappa \times 2000)$$

from which we find:

$$\kappa = -\frac{1}{2000} \ln \left(\frac{1000}{5 \times 10^6} \right) \approx 0.00425 \quad (\text{in years}).$$

Now, if a generation is 20 years, $\kappa \approx 20 \cdot 0.00425 = 0.085$ in generations. With the coalescent with recombination, time is scaled in units of $2N$ generations, so $\kappa = 2 \times N \times 0.0085 \approx 1700$, if N , the effective population size is 10 000.

8.2 The μ parameter for SNPs.

It could be difficult to choose a value for this parameter. As r is the probability that a recombination occurs in one sequence, μ is the probability to have a mutation in a sequence. The mutation rate should not be too low, because with the infinitely many sites models, a mutation could occur in a sequence if no other sequences share this mutation (a mutation happens only once for a given marker locus in the history of the sample).

8.3 The ReWeight (ξ) parameter. \star

The idea is that more the longer the sequence, more recombination events occur, and the larger is the number of steps to find the MRCA. So the ξ parameter acts on the probability of a Re event. In equation (6.9) of Larribe *et al.* (2002), just multiply the probability of Re by ξ , and divide the functional f in (6.10) by ξ for a recombination transition. The objective is to speed up the process, but of course, there is a price to pay. The recombination events are the most informative events used to localize the TIM, and when $\xi < 1$, less recombination events will occur.

8.4 The MaxIter parameter.

Mathematically, the number of ancestors follows a birth and death process, and this process converges. But some reconstructed graphs could be very long. It is then a good idea to anticipate such cases, and this option permits to stop

the reconstruction of a graph if the number of steps in a current graph is greater than `MaxIter`, and returns 0 for the functional value of this graph. To begin with, you should give a big value, as 10000, or larger depending on the data.

Please note that the information on the number of graphs aborted by the program is indicated in the output file by the `DropOut [0 0 0.001]`, for example, meaning that no reconstructed of graphs have been stopped when working on the first two intervals, and 0.1 % of the graphs have been stopped when working on the third interval.

8.5 How to combine results from different runs ?

The likelihood is, for K iterations:

$$L = \frac{1}{K} \sum_{i=1}^K \left[\prod_{\tau=0}^{\tau^*} f(\cdot) \right],$$

which is equivalent to:

$$\begin{aligned} L &= \frac{1}{K_1 + K_2} \sum_{i=1}^{K_1+K_2} \left[\prod_{\tau=0}^{\tau^*} f(\cdot) \right] \\ &= \frac{1}{K_1 + K_2} \sum_{i=1}^{K_1} \left[\prod_{\tau=0}^{\tau^*} f(\cdot) \right] + \frac{1}{K_1 + K_2} \sum_{i=K_1+1}^{K_1+K_2} \left[\prod_{\tau=0}^{\tau^*} f(\cdot) \right], \\ &= \frac{1}{K_1 + K_2} [K_1 L_1 + K_2 L_2] \end{aligned}$$

where L_1 is the likelihood over the first K_1 iterations, and L_2 is the likelihood over the last K_2 iterations. So, several computers could be used as long as the random seed is different for each simulation. Then, you can just combine the results using the above method, or by using the provided programs (`CombineS.exe` and `CombineM.exe`).

8.6 Trace of estimation

Two parameters could be used to follow estimation during the processing. These are `TraceRt` and `FreqTraceRt`. If `TraceRt` is 1 (default is 0), the trace option is enabled: so a graph will be added in the GnuPlot output which shows

shows the number of iterations of the process in the x axis, and in the y axis, shows the value of the log-likelihood and the maximum likelihood estimate for some value of the number of iterations. So we can “follow” the estimation. This is an exploratory option, which demands a lot of additional computing time.

8.7 Program errors and remarks.

It is possible that the program stops and outputs an error, if the input files are not exactly what the program expects. Avoid long file names, and special characters in file names. If you cannot find the error, start from the samples files provided with the program, and modify them. Ends of lines must be of UNIX type (even if you work in Windows); some free software could be used to obtain easily these ends of lines, such as MetaPad (<http://liquidninja.com/metapad/>), which is an improved version of Notepad. In the parameter file, no space should be added after the value of a parameter.

If you see some undocumented variables in the program or in the output, these are some options in development, and should not be considered.

MAPARG© (. /52_03 /52)

ResSp1

Parameters

Data File	peter.dat	n	10	d	6
$L - 1$	4	K	5m	ψ	500
Par. File	peter.par	r	0.000125	N_e	10000
ρ	5	θ	5	κ	0
μ_l	2.5e-05	Simple	0	Re Type	0
Seed	1	Can	[7 ; 77 ; 17]	Res. phase	-1

Results

r_T	0.00716367	Interval	2	Time/1M	0j 0h 40m 0s
% DropOut	[0;0;0]	CPU time	0j 0h 0m 11s	Real time	0j 0h 0m 12s

Approximate 95% CI [0.0030615;0.0098086]

ESS [6.8901 ; 1.2609 ; 2.8373]

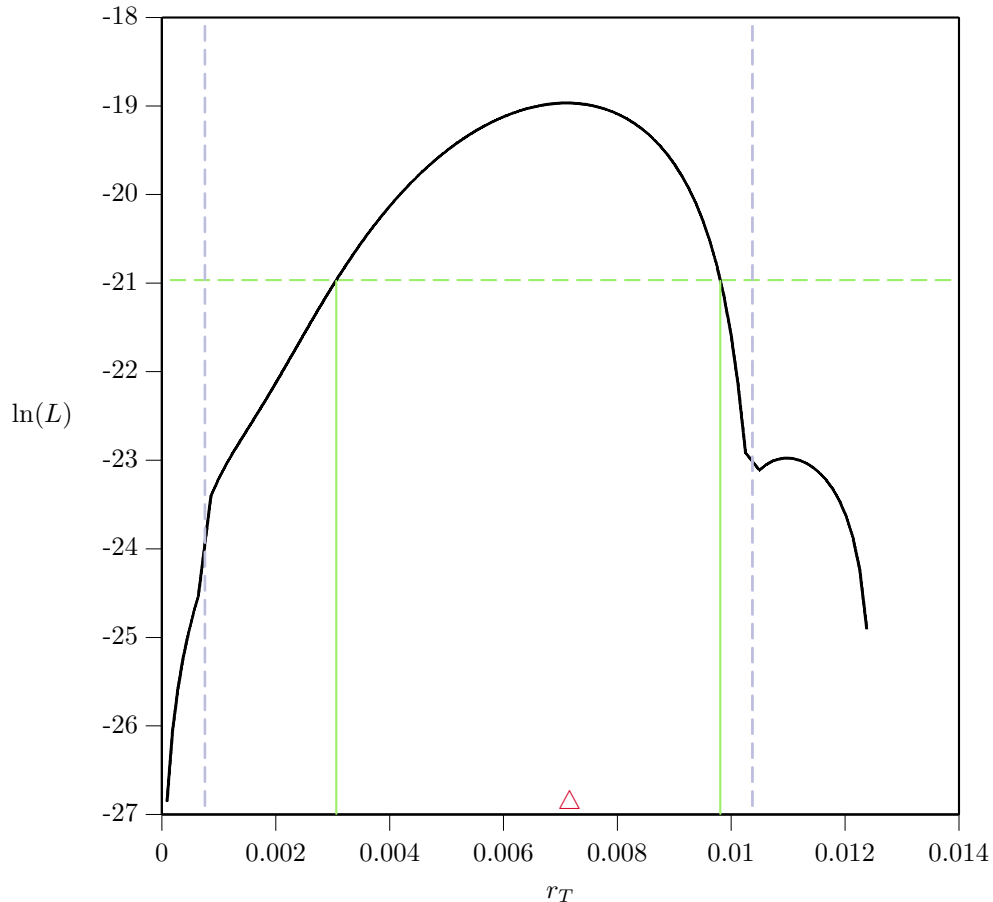
Nb of graphs really done [5000 ; 5000 ; 5000]

Nb of events [78.03±47.25 ; 33.38±7.986 ; 45.78±18.07]

Nb of Co. [41.02±23.63 ; 18.69±3.993 ; 24.89±9.037]

Nb of Mu. [5±0 ; 5±0 ; 5±0]

Nb of Re. [32.02±23.63 ; 9.689±3.993 ; 15.89±9.037]



Annexe C

Programme informatique

Nous présentons dans cette annexe le programme adapté aux SNPs. Celui développé pour le modèle microsatellite est similaire.

```
/**
** Develop by Fabrice Larribe.
** Contacs: larribe@dms.umontreal.ca
**          or lessards@dms.umontreal.ca
** This a beta version. Do not distribute.
** Only for SNPs.
**/

//#define Extnum 1 // If this line is in comment, the program do not use EXTNUM numbers ;

#include <iostream>      #include <iterator>      #include <deque>      #include <vector>
#include <math.h>        #include <ctime>          #include <algorithm>  #include <fstream>
#include <string>        #include <cstdio>        #include <cmath>      #include <stdio.h>
#include "MersenneTwister.h"
using namespace std;

#ifdef Extnum
    #include "real.h"
    #define EXTNUM extnum
#endif

#ifndef Extnum
    #include <iomanip>
    typedef long double longdouble ;
    #define EXTNUM longdouble
#endif

//#define DEBUG 1 ; // if this line is uncoment, the DEBUG mode
// is enable. Thsi means a trace of the progamm
// is output to file or to sreen.
```

```

// Definition of the differents type of vectors used ;
typedef deque<int> List;
typedef deque<float> ListF;
typedef deque<long double> ListD;
typedef deque<EXTNUM> ListE;
typedef vector<int> Vec;
typedef vector<Vec > VecOfVec;
typedef vector<EXTNUM> VecE;
typedef vector<short int> VecSI;
typedef deque<List > ListOfList;
typedef deque<ListF > ListOfListF;
typedef deque<ListD > ListOfListD;
typedef deque<unsigned long> ListUL;
typedef vector<double> VecD;
typedef vector<VecD > VecOfVecD;
typedef vector<VecE > VecOfVecE;
typedef vector<VecSI > VecOfVecSI;

// Trace route variables ;
List VTraceRtS ;
ListD VTraceRt , VTraceRtL ;
ListOfListD VVTraceRt , VVTraceRtL ;
ListOfList VVTraceRtS ;
int MaxDim = 0 ;
VecD VDim , VCR ;
List VTraceRtI ;
int FreqTraceRt = 1 ;
int TraceRt = 0;
long double last_rt_TraceRt = 0;
long double last_rtL_TraceRt = 0;

// Iterator ;
VecOfVec::iterator si , si1 , si2;
VecOfVecD::iterator sid , sid1 , sid2;
VecE::iterator sje ;
VecOfVecE::iterator sie ;
VecD::iterator sjr ;
Vec::iterator sj , sj1 , sj2;
VecSI::iterator sjk , sjk1 , sjk2 ;
VecOfVecSI::iterator sik ;
ListOfList::iterator i;
ListOfList::iterator i1 , i2;
List::iterator j , j2;
ListOfListD::iterator ir;
ListD::iterator jr;
ListE::iterator ie;
ListOfListD::iterator id;

// Backup and restore ;
int SimRestore , intervalRestore , InfoRestore = 0 ;
unsigned long RandomSeed = 0 ;
VecE LikERestore;
VecD VSNbRRestore , VS2NbRRestore , VSNbMRestore , VS2NbMRestore ,
      VSNbCRestore , VS2NbCRestore , VSNbERestore , VS2NbERestore ,
      VNbGraphDoneRestore ;
VecE SumWiRestore , SumWi2Restore ;

// Tex Output ;
VecD VX , VY , VI ;
int cpujours , cpuheures , cpuminutes , cpusecondes ,

```

```

    realjours , realheures , realminutes , realsecondes ;
int IntervalMax , Mjours , Mheures , Mminutes , Msecondes ;

short int T_j , T_k ;
VecD Prob , Prob0;
VecOfVecSI PEvent , PEvent0;
VecOfVec THap , TMu , TRe , TCoa , TAnc , TReR , TReL ;
Vec TNi , TNi0;
VecD Tb , FreqPop, FreqAA;
ListD Theta0 , Mu0, Dist0 , Dist ;
ListF Abandon ;
Vec VNbCan , VNprime , VNprime0 , Ta , VMutation , VMutation0 ;
List Trait ;
VecSI ID , ID0;
ListOfList Y , Y1;
ListUL LikSeed ;
VecE LikVal ;

// Likelihood variables ;
EXTNUM MinimumLik = 0 , LikP ;
ListE LikMax ;
VecE LikE , LikR , LikM , SumWi , SumWi2 , ESS ;
long double lambda = 1e300L ;

// Descriptives Statistics;
int NbR , NbM , NbC , NbE , NbGraphDone ;
VecD VSNbR , VS2NbR , VSNbM , VS2NbM , VSNbC , VS2NbC , VSNbE ,
VS2NbE , VNbGraphDone ;

// Resampling variables ;
List LikStep ;
ListD VRt ;
List VG ;
unsigned long GraphSeed ;
int R , replicate , NResamp = -1 , NResamp_i = -1 , NbGraphToKeep = -1 ,
NbGraphToKeep_i = -1 ;
int Phase = -1 , Phase0 = -1 , Max_pos = -1 , Nbackup , iterbackup , G , StepP ,
abackup;
Vec VNbSim , IDbackup , TNibackup , VMutationbackup , VNprimebackup ;
long double bbackup;
VecE ProdFEbackup;
VecOfVecSI PEventbackup ;
VecD Probbbackup ;
MTRand::uint32 randState[ MTRand::SAVE ] ;

// Files variables ;
char *par_file = "toto.par" , *dat_file , *rnd_file , *res_file , *program_name ;
char *res_file_i ; // Result File ask by the command line ;
char *rnd_file_i ; // Rnd File ask by the command line ;
string FileR ;

// Program / Options variables ;

```

```

const double version_programm = 52.03 ;

// Data variables ;
short int L ; // Nb of markers + Disease;
short int MarkerT = 1 ; //Marker Type: SNP = 1, Micro = 0;
bool RM = 0 ; // RecurrentMutation: 1 = Yes / 0 = No

// Others variables ;
float muinit = 0;
long double alpha , beta ;
int MeanNbSim , NbAncestors = 1 ;
string CpuTime , RealTime ;
VecD DrivingInitial , Driving ;
List SeqTmp1 , SeqTmp2;
short int Ti , Tj , Tk ; int n_i , n_j , n_k ;
int Ne , NbSim , NbCan , n , PrintNbSim = -1 , MaxIter, ChoosenEvent , debug =
0;
int K = 0 , N , N0, NT , NM , ni , nj , itmp, TypeEv , Choix , ExpLimit = 300, a
, a0 ;
unsigned long RandomSeedRestore = 0 ;
int ReType = 0 , proj_nbsim = 0 , NS , NS1 , NS2 ;
double kappa = 0.0 ;
long double ReWeight = 1.0 , rt_estimate , PopSize , rho, r , D , b , b0 ;
long double theta, SommeP , F_XY , LF_XY , c_time , lambda_t ;
bool VPS = 0 , converge = 1 , plot = 1 , TeXOutput = 1 , Simple = 0 , add_one ,
NotTheMRCA = 0;
EXTNUM MinLik , MaxLik ;
MTRand mtrand1 , mtrand2 , mtrand3;

//: C09:Cpptime.h
// A simple time class
#ifdef CPPTIME_H
#define CPPTIME_H
#include <ctime>
#include <cstring>
#include <sstream>

template <typename T>
string T_to_str(T arg)
{
    ostringstream oss;
    oss << arg;
    return oss.str();
}

//+-----+;
//|      SecondsToTime2()                               |;
//+-----+;
string SecondsToTime2(long int seconds){
    string Time;
    long int jours = (long int)((long double)(seconds / 86400) ) ;
    seconds -= jours * 86400 ;

```

```

    long int heures = (long int)((long double)(seconds / 3600) );
    seconds -= heures * 3600 ;
    long int minutes = (long int)((long double)(seconds / 60) );
    seconds -= minutes * 60 ;
    long int secondes = seconds ;
    Time += T_to_str(jours); Time += "j " ;
    Time += T_to_str(heures); Time += "h " ;
    Time += T_to_str(minutes); Time += "m " ;
    Time += T_to_str(secondes); Time += "s" ;
    return(Time);
}

class Time {
    time_t t;
    tm local;
    char asciiRep[26];
    unsigned char lflag, aflag;
    void updateLocal() {
        if(!lflag) {
            local = *localtime(&t);
            lflag++;
        }
    }
    void updateAscii() {
        if(!aflag) {
            updateLocal();
            strcpy(asciiRep, asctime(&local));
            aflag++;
        }
    }
public:
    Time() { mark(); }
    void mark() {
        lflag = aflag = 0;
        time(&t);
    }
    const char* ascii() {
        updateAscii();
        return asciiRep;
    }
    // Difference in seconds:
    int delta(Time* dt) const {
        return int(difftime(t, dt->t));
    }
    int daylightSavings() {
        updateLocal();
        return local.tm_isdst;
    }
    int dayOfYear() { // Since January 1
        updateLocal();
        return local.tm_yday;
    }
}

```

```

int dayOfWeek() { // Since Sunday
    updateLocal();
    return local.tm_wday;
}
int since1900() { // Years since 1900
    updateLocal();
    return local.tm_year;
}
int month() { // Since January
    updateLocal();
    return local.tm_mon;
}
int dayOfMonth() {
    updateLocal();
    return local.tm_mday;
}
int hour() { // Since midnight, 24-hour clock
    updateLocal();
    return local.tm_hour;
}
int minute() {
    updateLocal();
    return local.tm_min;
}
int second() {
    updateLocal();
    return local.tm_sec;
}
};
#endif // CPPTIME_H ///:~

#ifdef DEBUG
template <class T>
void PrintVV(vector<T> GListTmp, char *name){
    cout << "Liste \" << name << "\" (Taille \" << GListTmp.size() << ")\"n";
    typename vector<T>::iterator it;
    typename T::iterator jt;
    itmp = 0;
    for(it=GListTmp.begin(); it != GListTmp.end(); ++it){
        cout << " \" << itmp << ": \" ;
        itmp++;
        for(jt>(*it).begin(); jt != (*it).end(); ++jt)
            cout << *jt << " \" ;
        cout << "\"n";
    }
}

template <class T>
void PrintGL(deque<T> GListTmp, char *name){
    cout << "Liste \" << name << "\" (Taille \" << GListTmp.size() << ")\"n";
    typename deque<T>::iterator it;
    typename T::iterator jt;

```

```

    itmp = 0;
    for(it=GListTmp.begin(); it != GListTmp.end(); ++it){
        cout << " " << itmp << ": " ;
        itmp++;
        for(jt>(*it).begin(); jt != (*it).end(); ++jt)
            cout << *jt << " " ;
        cout << "\n";
    }
}

```

```

template <class T>
void Print2GL(deque<T > GListTmp, deque<T > ListCote){
    cout << "+ Data :\n" ;
    typename deque<T >::iterator it;
    typename deque<T >::iterator it2;
    typename T::iterator jt;
    typename T::iterator jt2;
    itmp = 0;
    it2 = ListCote.begin();
    for(it=GListTmp.begin(); it != GListTmp.end(); ++it){
        cout << " " << itmp << ": " ;
        itmp++;
        int indexbase = 0;
        for(jt>(*it).begin(); jt != (*it).end(); ++jt){
            ++indexbase;
            if(indexbase == 1) cout <<" [";
            cout << *jt << " " ;
            if(indexbase == 1) cout <<" ] ";
        }
        cout << "[ ";
        for(jt2>(*it2).begin(); jt2 != (*it2).end(); ++jt2)
            cout << *jt2 << " " ;
        cout << "]\n";
        ++ it2;
    }
}

```

```

void PrintPEvent() {
    int NbEv = PEvent.size();
    cout << "+ " << NbEv << " Events :\n";
    for(itmp = 0 ; itmp != NbEv ; ++itmp){
        if(itmp == 0){
            if(PEvent[itmp][0] == 1) cout <<" *Ci* ";
            if(PEvent[itmp][0] == 2) cout <<" -Cd- ";
            if(PEvent[itmp][0] == 3) cout <<" =Mu= ";
            if(PEvent[itmp][0] == 4) cout <<" ~Re~ ";
        }
        if(itmp != 0){
            if(PEvent[itmp][0] == 1 && PEvent[itmp-1][0] != PEvent[itmp][0] )
                cout <<"\n *Ci* ";
            if(PEvent[itmp][0] == 2 && PEvent[itmp-1][0] != PEvent[itmp][0] )

```



```

        cout << "\n -Cd- ";
        if(PEvent[itmp][0] == 3 && PEvent[itmp-1][0] != PEvent[itmp][0] )
        cout << "\n =Mu= ";
        if(PEvent[itmp][0] == 4 && PEvent[itmp-1][0] != PEvent[itmp][0] )
        cout << "\n ~Re~ ";
    }
    cout << " [" << itmp << "]" ";
    VecSI TEvent = PEvent[itmp] ;
    for(sjk=TEvent.begin()+1 ; sjk != TEvent.end() ; ++sjk) cout << *sjk << " " ;
    cout << "(" << Prob[itmp] / SommeP * 100 << ")" ;
}
cout << "] \n";
}

```

```

template <class T>
void PrintPL(T ListTmp, char *name){
    typename T::iterator j;
    cout << "Liste \" << name << "\" (Taille \" << ListTmp.size() << ")";
    cout << " [";
    for(j=ListTmp.begin(); j != ListTmp.end(); ++j){
        cout << " " << *j ;
    }
    cout << " ] \n";
}

```

```

#endif
template <class T>
void PrintV(T ListTmp, char *name){
    typename T::iterator j;
    cout << " " << name ;
    cout << " [";
    for(j=ListTmp.begin(); j != ListTmp.end(); ++j){
        cout << " " << *j ;
    }
    cout << " ] (" << ListTmp.size() << ") \n";
}

```

```

struct TTree{
    int n_b ;
    int n_e ;
    TTree* zeroTree ;
    TTree* oneTree ;
    TTree* nineTree ;
}
FirstNode;
TTree* m_current ;

void del_seq_tree_b(int Is , TTree *top){
    Vec Seq = THap[Is] ;
    Vec::iterator iL;
    int multi = 1 ;
}

```

```

m_current = top ;
for (iL=Seq.begin() ; iL != Seq.end()-1; ++iL){
    if(*iL == 0){
        TTree* pnewzero ;
        if(m_current->zeroTree == NULL){
            pnewzero = new TTree();
            pnewzero->zeroTree = NULL ;
            pnewzero->oneTree = NULL ;
            pnewzero->nineTree = NULL ;
            pnewzero->n_b = 0 ;
            pnewzero->n_e = 0 ;
            m_current->zeroTree = pnewzero ;
        }
        else pnewzero = m_current->zeroTree ;
        m_current = pnewzero ;
        m_current->n_b -= multi ;
    }
    else if(*iL == 1){
        TTree* pnewone ;
        if(m_current->oneTree == NULL){
            pnewone = new TTree();
            pnewone->zeroTree = NULL ;
            pnewone->oneTree = NULL ;
            pnewone->nineTree = NULL ;
            pnewone->n_b = 0 ;
            pnewone->n_e = 0 ;
            m_current->oneTree = pnewone ;
        }
        else pnewone = m_current->oneTree ;
        m_current = pnewone ;
        m_current->n_b -= multi ;
    }
    else if(*iL == 9){
        TTree* pnewnine ;
        if(m_current->nineTree == NULL){
            pnewnine = new TTree();
            pnewnine->zeroTree = NULL ;
            pnewnine->oneTree = NULL ;
            pnewnine->nineTree = NULL ;
            pnewnine->n_b = 0 ;
            pnewnine->n_e = 0 ;
            m_current->nineTree = pnewnine ;
        }
        else pnewnine = m_current->nineTree ;
        m_current = pnewnine ;
        m_current->n_b -= multi ;
    }
}
}

void delete_node(TTree *node){
    TTree* pnew;

```

```

    pnew = node ;
    if(node->zeroTree != NULL) delete_node(node->zeroTree) ;
    if(node->oneTree != NULL) delete_node(node->oneTree) ;
    if(node->nineTree != NULL) delete_node(node->nineTree) ;
    node->zeroTree = NULL ;
    node->oneTree = NULL ;
    node->nineTree = NULL ;
    delete node ;
}

void delete_tree(){
    m_current= &FirstNode ;
    if(m_current->zeroTree) delete_node(m_current->zeroTree) ;
    if(m_current->oneTree ) delete_node(m_current->oneTree) ;
    if(m_current->nineTree) delete_node(m_current->nineTree) ;
    m_current->zeroTree = NULL ;
    m_current->oneTree = NULL ;
    m_current->nineTree = NULL ;
}

void add_seq_tree_b(int Is , TTree *top){
    Vec::iterator iL ;
    Vec Seq = THap[Is] ;
    int multi ;
    if(add_one == 0) multi = TNi[Is] ;
    else multi = 1 ;
    m_current= top ;
    for(iL=Seq.begin() ; iL != Seq.end()-1; ++iL){
        if(*iL == 0){
            TTree* pnewzero ;
            if(m_current->zeroTree == NULL){
                pnewzero = new TTree() ;
                pnewzero->zeroTree = NULL ;
                pnewzero->oneTree = NULL ;
                pnewzero->nineTree = NULL ;
                pnewzero->n_b = 0 ;
                pnewzero->n_e = 0 ;
                m_current->zeroTree = pnewzero ;
            }
            else pnewzero = m_current->zeroTree ;
            m_current = pnewzero ;
            m_current->n_b += multi ;
        }
        if(*iL == 1){
            TTree* pnewone ;
            if(m_current->oneTree == NULL){
                pnewone = new TTree() ;
                pnewone->zeroTree = NULL ;
                pnewone->oneTree = NULL ;
                pnewone->nineTree = NULL ;
                pnewone->n_b = 0 ;
                pnewone->n_e = 0 ;
            }
        }
    }
}

```

```

        m_current->oneTree = pnewone ;
    }
    else pnewone = m_current->oneTree ;
    m_current = pnewone ;
    m_current->n_b += multi ;
}
else if(*iL == 9){
    TTree* pnewnine ;
    if(m_current->nineTree == NULL){
        pnewnine = new TTree() ;
        pnewnine->zeroTree = NULL ;
        pnewnine->oneTree = NULL ;
        pnewnine->nineTree = NULL ;
        pnewnine->n_b = 0 ;
        pnewnine->n_e = 0 ;
        m_current->nineTree = pnewnine ;
    }
    else pnewnine = m_current->nineTree ;
    m_current = pnewnine ;
    m_current->n_b += multi ;
}
}
}

void add_seq_tree_e(int Is , TTree *top){
    Vec::iterator iL ;
    Vec Seq = THap[Is] ;
    int multi ;
    if(add_one == 0) multi = TNi[Is] ;
    else multi = 1 ;
    m_current= top ;
    m_current= top ;
    for(iL=Seq.end()-1 ; iL > Seq.begin() ; --iL){
        if(*iL == 0){
            TTree* pnewzero ;
            if(m_current->zeroTree == NULL){
                pnewzero = new TTree() ;
                pnewzero->zeroTree = NULL ;
                pnewzero->oneTree = NULL ;
                pnewzero->nineTree = NULL ;
                pnewzero->n_b = 0 ;
                pnewzero->n_e = 0 ;
                m_current->zeroTree = pnewzero ;
            }
            else pnewzero = m_current->zeroTree ;
            m_current = pnewzero ;
            m_current->n_e += multi ;
        }
        else if(*iL == 1){
            TTree* pnewone ;
            if(m_current->oneTree == NULL){
                pnewone = new TTree() ;

```

```

        pnwone->zeroTree = NULL ;
        pnwone->oneTree = NULL ;
        pnwone->nineTree = NULL ;
        pnwone->n_b = 0 ;
        pnwone->n_e = 0 ;
        m_current->oneTree = pnwone ;
    }
    else pnwone = m_current->oneTree ;
    m_current = pnwone ;
    m_current->n_e += multi ;
}
else if(*iL == 9){
    TTree* pnwnine ;
    if(m_current->nineTree == NULL){
        pnwnine = new TTree() ;
        pnwnine->zeroTree = NULL ;
        pnwnine->oneTree = NULL ;
        pnwnine->nineTree = NULL ;
        pnwnine->n_b = 0 ;
        pnwnine->n_e = 0 ;
        m_current->nineTree = pnwnine ;
    }
    else pnwnine = m_current->nineTree ;
    m_current = pnwnine ;
    m_current->n_e += multi ;
}
}
}

void del_seq_tree_e(int Is , TTree *top){
    Vec::iterator iL;
    Vec Seq = THap[Is] ;
    int multi = 1 ;
    m_current= top ;
    m_current= top ;
    for(iL=Seq.end()-1 ; iL > Seq.begin() ; --iL){
        if(*iL == 0){
            TTree* pnwzero ;
            if(m_current->zeroTree == NULL){
                pnwzero = new TTree() ;
                pnwzero->zeroTree = NULL ;
                pnwzero->oneTree = NULL ;
                pnwzero->nineTree = NULL ;
                pnwzero->n_b = 0 ;
                pnwzero->n_e = 0 ;
                m_current->zeroTree = pnwzero ;
            }
            else pnwzero = m_current->zeroTree ;
            m_current = pnwzero ;
            m_current->n_e -= multi ;
        }
        else if(*iL == 1){

```

```

    TTree* pnewone ;
    if(m_current->oneTree == NULL){
        pnewone = new TTree();
        pnewone->zeroTree = NULL ;
        pnewone->oneTree = NULL ;
        pnewone->nineTree = NULL ;
        pnewone->n_b = 0 ;
        pnewone->n_e = 0 ;
        m_current->oneTree = pnewone ;
    }
    else pnewone = m_current->oneTree ;
    m_current = pnewone ;
    m_current->n_e -= multi ;
}
else if(*iL == 9){
    TTree* pnewnine ;
    if(m_current->nineTree == NULL){
        pnewnine = new TTree();
        pnewnine->zeroTree = NULL ;
        pnewnine->oneTree = NULL ;
        pnewnine->nineTree = NULL ;
        pnewnine->n_b = 0 ;
        pnewnine->n_e = 0 ;
        m_current->nineTree = pnewnine ;
    }
    else pnewnine = m_current->nineTree ;
    m_current = pnewnine ;
    m_current->n_e -= multi ;
}
}

void add_seq_tree(int Is){
    add_seq_tree_b(Is , &FirstNode);
    add_seq_tree_e(Is , &FirstNode);
}

void del_seq_tree(int Is){
    del_seq_tree_b(Is , &FirstNode);
    del_seq_tree_e(Is , &FirstNode);
}

int how_much_seq_b(Vec Seq){
    m_current = &FirstNode;
    Vec::iterator iL;
    for(iL=Seq.begin() ; iL != Seq.end(); ++iL){
        if(*iL == 0) m_current = m_current->zeroTree ;
        if(*iL == 1) m_current = m_current->oneTree ;
        if(*iL == 9) m_current = m_current->nineTree ;
    }
    return(m_current->n_b);
}

```

```

int how_much_seq_e(Vec Seq){
    m_current = &FirstNode;
    Vec::iterator iL;
    for(iL=Seq.end()-1 ; iL >= Seq.begin(); --iL){
        if(*iL == 0) m_current = m_current->zeroTree ;
        if(*iL == 1) m_current = m_current->oneTree ;
        if(*iL == 9) m_current = m_current->nineTree ;
    }
    return(m_current->n_e);
}

Vec possible_ancestor_r(int Is ){
    Vec PAR;
    Vec Seq = THap[Is];
    m_current = &FirstNode ;
    Vec::iterator iL;
    for(iL=Seq.begin() ; iL != Seq.end()-1; ++iL){
        if(*iL == 0) m_current = m_current->zeroTree ;
        if(*iL == 1) m_current = m_current->oneTree ;
        if(*iL == 9) m_current = m_current->nineTree ;
        PAR.push_back(m_current->n_b);
    }
    return(PAR);
}

Vec possible_ancestor_l(int Is ){
    Vec PAR ;
    Vec Seq = THap[Is];
    m_current = &FirstNode ;
    Vec::iterator iL;
    for(iL=Seq.end()-1 ; iL > Seq.begin(); --iL){
        if(*iL == 0) m_current = m_current->zeroTree ;
        if(*iL == 1) m_current = m_current->oneTree ;
        if(*iL == 9) m_current = m_current->nineTree ;
        PAR.push_back(m_current->n_e);
    }
    return(PAR);
}

int arondi_di(double x) {
    return (int)(x > 0.0 ? x + 0.5 : x - 0.5);
}

int arondi(double x) {
    return (int)(x > 0.0 ? x + 0.5 : x - 0.5);
}

char* ConvertSC(string mot){ // Convert a String to Char ;
    char *tmp = new char[mot.length()+1];
    mot.copy(tmp, mot.length());
    tmp[mot.length()+1] = 0;
}

```

```

    return(tmp);
}

char* ConvertSC2(string mot){ // Convert a String to Char ;
    char *tmp = new char[mot.length()+0];
    mot.copy(tmp, mot.length());
    tmp[mot.length()+0] = 0;
    return(tmp);
}

int f_MaxD(VecD Liste){
    double Max_value = *(Liste.begin()+0);
    int IndexMax = 0 , itmp = 0;
    for(sjr=Liste.begin()+1 ; sjr != Liste.end() ; ++sjr){
        ++ itmp ;
        if(*sjr > Max_value){
            IndexMax = itmp ;
            Max_value = *sjr ;
        }
    }
    return(IndexMax);
}

int f_MinD(VecD Liste){
    double Min_value = *(Liste.begin()+0);
    int IndexMax = 0 , itmp = 0;
    for(sjr=Liste.begin()+1 ; sjr != Liste.end() ; ++sjr){
        ++ itmp ;
        if(*sjr < Min_value){
            IndexMax = itmp ;
            Min_value = *sjr ;
        }
    }
    return(IndexMax);
}

int f_Min(VecE Liste){
    EXTNUM Min_value = *(Liste.begin()+0);
    int IndexMax = 0 , itmp = 0;
    for(sje=Liste.begin()+1 ; sje != Liste.end() ; ++sje){
        ++ itmp ;
        if(*sje < Min_value){
            IndexMax = itmp ;
            Min_value = *sje ;
        }
    }
    return(IndexMax);
}

int f_Max(VecE Liste){
    EXTNUM Max_value = *(Liste.begin()+0);
    int IndexMax = 0 , itmp = 0;

```



```

    for(sje=Liste.begin()+1 ; sje != Liste.end() ; ++sje){
        ++ itmp ;
        if(*sje > Max_value){
            IndexMax = itmp ;
            Max_value = *sje ;
        }
    }
    return(IndexMax);
}

EXTNUM atofd(string mot){
    string first , second;
    int positif = 1 , avant = 1;
    for(unsigned int j1=0 ; j1!= mot.size() ; ++j1){
        if(mot[j1] == 'e' || mot[j1] == 'E'){
            if(mot[j1+1] == '-') positif = 0;
            ++j1;
            avant = 0;
        }
        if(avant == 1) first = first + mot[j1];
        if(avant == 0) second += mot[j1];
    }
    EXTNUM x = (EXTNUM)(atof(first.c_str()));
    int y = atoi(ConvertSC2(second));
    if(y < 0) {
        positif = 0 ;
        y *= - 1;
    }
    if(y<0) y *= -1 ;
    EXTNUM z = x ;
    for(int j2=0 ; j2 != y ; ++j2){
        if(positif == 1 ) z *= 10;
        if(positif == 0 ) z /= 10;
    }
    return(z);
}

// +-----+
// | void PrintData() |
// +-----+
void PrintData(){
    cout<<" + Haplotypes:"<<endl;
    for(sjk=ID.begin() ; sjk != ID.end(); ++sjk){
        if(*sjk<=9) cout <<" [ "<<*sjk<<" ] ";
        if(*sjk<=99 && *sjk>9) cout <<" [ "<<*sjk<<" ] ";
        if(*sjk>99) cout <<" [ "<<*sjk<<" ] ";
        cout<<TNi[*sjk]<<" [ ";
        for(int itmp = 0 ; itmp != L ; ++itmp) cout << THap[*sjk] [itmp] <<" ";
        cout<<" ] "<< TAnc[*sjk] [0] <<" : "<< TAnc[*sjk] [1] <<" "<<endl;
    }
}

```

```

// +-----+
// | void SaveIntermediateResult() |
// +-----+
void SaveIntermediateResult(int interval){

    string FileRND = (string)(rnd_file) + ".rnd";
    const char* rnd_file = FileRND.c_str() ;
    ofstream outR;
    outR.open(rnd_file, ios::app);
    if(!outR){
        char *rnd_back="MapArgDefault.rnd";
        cerr << " + Rnd file could not be opened for writing.\n";
        cerr << " + Rnd File is then: MapArgDefault.rnd.\n";
        ofstream outR(rnd_back,ios::app);
    }

    // Save graph seed for resampling ;
    if(Phase == 0){
        for(int itemp=0 ; itemp != NbGraphToKeep ; itemp++){
            if(LikStep[itemp] > 0 ){
                #ifdef Extnum
                    outR << interval<<" " << LikMax[itemp] << " " << LikStep[itemp] <<"
" << LikSeed[itemp] << "\n";
                #endif
                #ifndef Extnum
                    outR << interval<<" " << LikMax[itemp]/lambda << " " << LikStep[itemp]
<<" " << LikSeed[itemp] << "\n";
                #endif
            }
        }
    }
    outR.close();

    // Save estimate of r_t for trace route;
    if(TraceRt == 1){
        string FileLik = FileR + ".lik";
        const char* lik_file = FileLik.c_str();
        ofstream outl(lik_file,ios::out);
        if(!outl){
            char *lik_back = "MapArgDefault.lik";
            cerr << " + LIK file could not be opened for writing.\n";
            cerr << " + Result File is then: MapArgDefault.lik.\n";
            ofstream outl(lik_back,ios::out);
            exit(5);
        }
        int index_sim = 0;
        int index_pos = 0;
        for(jr=VTraceRt.begin() ; jr!= VTraceRt.end(); ++jr){
            index_sim += FreqTraceRt ;
            outl << index_sim << " " << *jr << " " << VTraceRtL[index_pos]<<"\n";
            cout << index_sim << " " << *jr << " " << VTraceRtL[index_pos]<<"\n";
        }
    }
}

```

```

        ++ index_pos ;
    }
}

// -----+
// | void Backup() |
// -----+
void Backup(int Sim, int interval ){

    string FileBackup = FileR + ".bck";
    const char* backup_file = FileBackup.c_str();
    ofstream outb(backup_file, ios::out);
    if(!outb){
        char *bck_back = "MapArgDefault.bck";
        cerr << " + Backup File could not be opened for writing.\n";
        cerr << " + Backup File is then: MapArgDefault.bck.\n";
        ofstream outb(bck_back, ios::out);
    }
    outb << "interval " << interval << endl;
    outb << "Sim " << Sim << endl;
    outb << "RandomSeed " << RandomSeed << endl;
    outb << "Like";
    for(sje=LikE.begin() ; sje!=LikE.end(); ++sje)
    outb<<" "<<*sje;
    outb<<endl;
    outb << "VNbGraphDone";
    for(sjr=VNbGraphDone.begin() ; sjr!=VNbGraphDone.end(); ++sjr)
    outb<<" "<<*sjr;
    outb<<endl;
    outb << "VSNbR";
    for(sjr=VSNbR.begin() ; sjr!=VSNbR.end(); ++sjr)
    outb<<" "<<*sjr;
    outb<<endl;
    outb << "VS2NbR";
    for(sjr=VS2NbR.begin() ; sjr!=VS2NbR.end(); ++sjr)
    outb<<" "<<*sjr;
    outb<<endl;
    outb << "VSNbM";
    for(sjr=VSNbM.begin() ; sjr!=VSNbM.end(); ++sjr)
    outb<<" "<<*sjr;
    outb<<endl;
    outb << "VS2NbM";
    for(sjr=VS2NbM.begin() ; sjr!=VS2NbM.end(); ++sjr)
    outb<<" "<<*sjr;
    outb<<endl;
    outb << "VSNbE";
    for(sjr=VSNbE.begin() ; sjr!=VSNbE.end(); ++sjr)
    outb<<" "<<*sjr;
    outb<<endl;
    outb << "VS2NbE";
    for(sjr=VS2NbE.begin() ; sjr!=VS2NbE.end(); ++sjr)

```

```

    outb<<" "<<*sjr;
    outb<<endl;
    outb << "VSNbC";
    for (sjr=VSNbC.begin() ; sjr!=VSNbC.end(); ++sjr)
    outb<<" "<<*sjr;
    outb<<endl;
    outb << "VS2NbC";
    for (sjr=VS2NbC.begin() ; sjr!=VS2NbC.end(); ++sjr)
    outb<<" "<<*sjr;
    outb<<endl;
    outb << "SumWi";
    for (sje=SumWi.begin() ; sje!=SumWi.end(); ++sje)
    outb<<" "<<*sje;
    outb<<endl;
    outb << "SumWi2";
    for (sje=SumWi2.begin() ; sje!=SumWi2.end(); ++sje)
    outb<<" "<<*sje;
    outb<<endl;
    string AleaBackUp = FileR + ".ran";
    const char* backup_Alea = AleaBackUp.c_str();
    ofstream stateOut(backup_Alea,ios::out);
    if( stateOut ) {
    stateOut << mtrand2;
    stateOut.close();
    }
}

// +-----+
// | void Restore() |
// +-----+
void Restore(){
    string AleaBackUp = FileR + ".ran";
    const char* backup_Alea = AleaBackUp.c_str();
    ifstream stateIn(backup_Alea,ios::in);
    if( stateIn ){
    stateIn >> mtrand2;
    stateIn.close();
    }

    string FileBackUp = FileR + ".bck";
    const char* backup_file = FileBackUp.c_str();
    ifstream outb(backup_file);
    if(!outb){
    char *bck_back = "MapArgDefault.bck";
    cerr << " + Restore File could not be opened for writing.\n";
    cerr << " + Restore File is then: MapArgDefault.bck.\n";
    ifstream outb(bck_back,ios::in);
    }
    string buf;
    while(getline(outb,buf) ){
    string::const_iterator cp = buf.begin();
    int cnt = 0;

```

```

int cas = 0;
while(cp != buf.end()){
    while(*cp == ' ') cp ++;
    if(*cp ){
        cnt++;
        string mot;
        while(*cp != ' ' && cp != buf.end()){
            mot += *cp;
            cp++;
        }
        if(cnt == 1){
            if(mot == "interval") cas = 1;
            if(mot == "Sim") cas = 2;
            if(mot == "LikE") cas = 3;
            if(mot == "RandomSeed") cas = 4;
            if(mot == "VNbGraphDone") cas = 5;
            if(mot == "VSNbR") cas = 6;
            if(mot == "VSNbM") cas = 7;
            if(mot == "VSNbE") cas = 8;
            if(mot == "VSNbC") cas = 9;
            if(mot == "VS2NbR") cas = 10;
            if(mot == "VS2NbM") cas = 11;
            if(mot == "VS2NbE") cas = 12;
            if(mot == "VS2NbC") cas = 13;
            if(mot == "SumWi") cas = 14;
            if(mot == "SumWi2") cas = 15;
        }
        if(cnt > 1){
            if(cas == 1) intervalRestore = atoi( mot.c_str() );
            if(cas == 2) SimRestore = atoi( mot.c_str() );
            if(cas == 3) LikERestore.push_back( atolfd( mot) );
            if(cas == 4) RandomSeedRestore = strtoul(
ConvertSC(mot), NULL, 10);
            if(cas == 5) VNbGraphDoneRestore.push_back( atolfd( mot) );
            if(cas == 6) VSNbRRestore.push_back( atolfd( mot) );
            if(cas == 7) VSNbMRestore.push_back( atolfd( mot) );
            if(cas == 8) VSNbERestore.push_back( atolfd( mot) );
            if(cas == 9) VSNbCRestore.push_back( atolfd( mot) );
            if(cas == 10) VS2NbRRestore.push_back( atolfd( mot) );
            if(cas == 11) VS2NbMRestore.push_back( atolfd( mot) );
            if(cas == 12) VS2NbERestore.push_back( atolfd( mot) );
            if(cas == 13) VS2NbCRestore.push_back( atolfd( mot) );
            if(cas == 14) SumWiRestore.push_back( atolfd(mot) );
            if(cas == 15) SumWi2Restore.push_back( atolfd(mot) );
        }
    }
}
}
outb.close();

LikE.assign(LikERestore.begin(), LikERestore.end());
RandomSeed = RandomSeedRestore ;

```

```

VNBGraphDone.assign(VNBGraphDoneRestore.begin(), VNBGraphDoneRestore.end());
VSNbR.assign(VSNbRRestore.begin(), VSNbRRestore.end());
VSNbM.assign(VSNbMRestore.begin(), VSNbMRestore.end());
VSNbE.assign(VSNbERestore.begin(), VSNbERestore.end());
VSNbC.assign(VSNbCRestore.begin(), VSNbCRestore.end());

VS2NbR.assign(VS2NbRRestore.begin(), VS2NbRRestore.end());
VS2NbM.assign(VS2NbMRestore.begin(), VS2NbMRestore.end());
VS2NbE.assign(VS2NbERestore.begin(), VS2NbERestore.end());
VS2NbC.assign(VS2NbCRestore.begin(), VS2NbCRestore.end());

SumWi.assign(SumWiRestore.begin(), SumWiRestore.end());
SumWi2.assign(SumWi2Restore.begin(), SumWi2Restore.end());

}

// +-----+
// | void WritePlot() |
// +-----+
void WritePlot(Vec VNBcan , ListD Dist ){

    string FileCMD = FileR + ".plt";
    const char* cmd_file = FileCMD.c_str() ;
    string FileDON = FileR + ".don";
    const char* don_file = FileDON.c_str() ;
    string FileLIK = FileR + ".lik";
    const char* lik_file = FileLIK.c_str() ;

    if(TraceRt == 1){
        ofstream outb(lik_file, ios::out);
        if(!outb){
            char *lik_back = "MapArgDefault.lik";
            cerr << " + Result File could not be opened for writing.\n";
            cerr << " + Result File is then: MapArgDefault.lik.\n";
            ofstream outb(lik_back, ios::out);
        }
        int index_pos = 0;
        for(jr=VTraceRtL.begin() ; jr!= VTraceRtL.end(); ++jr){
            outb << VTraceRtS[index_pos] << " " << VCR[VTraceRtI[index_pos]] << " "
                << VTraceRtL[index_pos] << "\n";
            ++ index_pos ;
        }
        outb.close();
    }

    ofstream outc(cmd_file, ios::out);
    if(!outc){
        char *cmd_back = "MapArgDefault.plt";
        cerr << " + Plot file could not be opened for writing.\n";
        cerr << " + Plot file is then: MapArgDefault.plt.\n";
        ofstream outc(cmd_back, ios::out);
    }
}

```

```

}
char *prefix = " ";
float K2 = NbSim;
if(K2>=1000000){
    K2 /= 1000000;
    prefix = "M";
}
else if(K2>=1000){
    K2 /= 1000;
    prefix = "m";
}
outc <<"# Command file for GNU PLOT"<<endl;
outc <<"set label 1 \"[K=\"<<K2<<prefix<<"] \" at screen 0.25, screen 0.9"<<endl;
outc <<"set label 2 \"[rt = \"<<rt_estimate<<\"] \" at screen 0.25, screen
0.85"<<endl;
outc <<"set grid"<<endl;
outc <<"set ylabel 'ln(L)'"<<endl;
outc <<"set xlabel 'r_T'"<<endl;
outc <<"plot \"<<don_file<<\" using ($3*100):6 with lines"<<endl;
outc <<"pause -1"<<endl;
outc <<"set ylabel 'L'"<<endl;
outc <<"f(x) = 0.1466"<<endl;
outc <<"plot \"<<don_file<<\" using ($3*100):4 with lines, f(x)"<<endl;
outc <<"pause -1"<<endl;
if(TraceRt == 1){
    outc << "set grid"<<endl;
    outc <<"set pointsize .1"<<endl;
    outc <<"set y2tics"<<endl;
    outc << "plot \"<<lik_file<<\" using 1:2 axes x1y1 with lines , \"<<lik_file
<<\" using 1:3 axes x1y2 with lines"<<endl;
    outc <<"pause -1"<<endl;
}
outc <<"# end"<<endl;
outc.close();

ofstream outd(don_file, ios::out);
if(!outd){
char *don_back = "MapArgDefault.don";
cerr << " + Plot (don) file could not be opened for writing.\n";
cerr << " + Plot file is then: MapArgDefault.don.\n";
ofstream outd(don_back, ios::out);
}
outd << "# Can Int    Coordinate    Rel Lik    Like    LLE
"<<endl;

int CCand = 0 ;
long double rglobal = 0 ;
for(unsigned int itemp3 = 1 ; itemp3 != Dist.size() + 1 ; ++ itemp3){
    double runit = Dist[itemp3-1] / (VNbCan[itemp3-1] + 1);
    for(int itmp2 =1; itmp2 != VNbCan[itemp3-1] +1; ++ itmp2){
        rglobal += runit ;
        outd.width(3);
    }
}

```

```

        outd << CCand+1 ;
        outd.width(3);
        outd << itemp3 ;
        outd.width(14);
        outd << rglobal ;
        outd.width(14);
        outd << LikR[CCand] ;
        outd.width(14);
        outd << LikE[CCand] ;
        outd.width(14);
        #ifdef Extnum
        outd << log(LikE[CCand]) << "\n";
        #endif
        #ifndef Extnum
        outd << log(LikE[CCand]) - log(lambda) << "\n";
        #endif
        CCand += 1 ;
    }
    rglobal += runit ;
}
outd << "\n";
outd.close();
}

```

```

void f(int l, Vec Seq){
    Vec Seq1;
    Vec Seq2;
    Vec Seq3;
    Seq1.assign(Seq.begin(), Seq.end());
    Seq2.assign(Seq.begin(), Seq.end());
    Seq3.assign(Seq.begin(), Seq.end());
    Seq1.push_back(0);
    Seq2.push_back(1);
    Seq3.push_back(9);
    if(l > 1) f(l-1, Seq1);
    if(l > 1) f(l-1, Seq2);
    if(l > 1) f(l-1, Seq3);
    if(l==1) THap.push_back(Seq1);
    if(l==1) THap.push_back(Seq2);
    if(l==1) THap.push_back(Seq3);
}

```

```

Vec f_ancestral(int L, Vec SeqTmp){
    Vec ListTmp;
    int MinCheck = 0, MinR = 0, MaxCheck = 0, MaxR = 0;
    int IM = 1;
    sj = SeqTmp.begin();
    while(MinCheck == 0 && IM <= L+1){
        if(MinCheck == 0 && *sj != 9){
            MinCheck = 1;
            MinR = IM ;
        }
    }
}

```



```

        ++ IM;
        ++ sj;
    }
    IM = L+1 ;
    sj = SeqTmp.end()-1;
    while(MaxCheck == 0 && IM>= 1){
        if(MaxCheck == 0 && *sj != 9 ){
            MaxCheck = 1;
            MaxR = IM ;
        }
        --IM;
        --sj;
    }
    ListTmp.push_back(MinR);
    ListTmp.push_back(MaxR);
    return(ListTmp);
}

// +-----+
// | int SeqPosL() |
// +-----+
int SeqPosL(int Ti){
    int spot = PEvent[ChoosenEvent][2] ;
    int NPoss = 0 , Test ; List Seqs;
    int Is = -1 , Pi = -1;
    si1 = THap.begin() + Ti;
    for(sjk=ID.begin() ; sjk!= ID.end() ; ++sjk){
        ++Is;
        if( *sjk == Ti) Pi = Is;
        si2 = THap.begin() + *sjk ;
        Test = 0;
        if( equal( (*si1).begin(), (*si1).begin()+spot, (*si2).begin() ) == 1 ){
            NPoss += TNi[*sjk] ;
            Test = 1;
        }
        Seqs.push_back( TNi[*sjk] * Test );
    }
    Seqs[Pi] = 0 ; NPoss -= TNi[Ti];
    if(NPoss == 0) return(-1);
    long double Alea = 1 ;
    if(NPoss > 1) Alea = rand()%(NPoss-1) + 1;
    int Pj = -1; int Somme = 0; j=Seqs.begin(); short int CS = -1 ;
    while(Pj == -1){
        CS ++; Somme += *j;
        if(Alea <= Somme) Pj = CS;
        ++j;
    }
    return( ID[Pj] );
}

// +-----+
// | int SeqPosR() |

```

```

// -----+
int SeqPosR(int Ti){
    int spot = PEvent[ChoosenEvent][2] ;
    int NPoss = 0 , Test ; List Seqs;
    int Is = -1 , Pi = -1;
    si1 = THap.begin() + Ti;
    for(sjk=ID.begin() ; sjk!= ID.end() ; ++sjk){
        ++Is;
        if( *sjk == Ti) Pi = Is;
        si2 = THap.begin() + *sjk ;
        Test = 0;
        if( equal( (*si1).begin()+spot, (*si1).end(), (*si2).begin()+spot ) == 1 ){
            NPoss += TNi[*sjk] ;
            Test = 1;
        }
        Seqs.push_back( TNi[*sjk] * Test );
    }
    Seqs[Pi] = 0 ; NPoss -= TNi[Ti] ;
    if(NPoss == 0) return(-1);
    long double Alea = 1 ;
    if(NPoss > 1) Alea = rand()%(NPoss-1) + 1;
    int Pk = -1; int Somme = 0; j=Seqs.begin(); short int CS = -1 ;
    while(Pk == -1){
        CS ++; Somme += *j;
        if(Alea <= Somme) Pk = CS;
        ++j;
    }
    return( ID[Pk] );
}

// -----+
// | int NumSeq() |
// -----+
short int NumSeq(Vec SeqTmp){
    short int NSeq = 0, TSeq = -1;
    VecOfVec::iterator si=THap.begin();
    while(si != THap.end()){
        if( equal( (*si).begin(), (*si).end(), SeqTmp.begin() ) == 1 ){
            TSeq = NSeq;
            break ;
        }
        ++ si ;
        ++ NSeq ;
    }
    return(TSeq);
}

// -----+
// | void CreateTableTB() |
// -----+
void CreateTableTB(){
    Tb.clear();
}

```

```

int IsT = -1;
for(si=THap.begin() ; si!= THap.end() ; ++ si){
    ++IsT;
    int MinR = (int)*((*(TAnc.begin()+IsT)).begin()+0);
    int MaxR = (int)*((*(TAnc.begin()+IsT)).begin()+1);
    float ck = 0;
    for(int itmp = MinR ; itmp <= MaxR -1; ++itmp){
        ck += *(Dist.begin()+ itmp -1);
    }
    Tb.push_back(ck);
}
}

// -----+
// | void CreateTableFreq() |
// -----+
void CreateTableFreq(){
    FreqPop.clear();
    Vec Tmp;
    for(si=THap.begin() ; si!= THap.end() ; ++ si){
        Tmp = *si;
        double prob = 1 ;
        for(short int IM=0 ; IM <= L-1 ; ++IM){
            if(Tmp[IM] == 0) prob *= FreqAA[IM];
            if(Tmp[IM] == 1) prob *= 1 - FreqAA[IM];
        }
        FreqPop.push_back(prob);
    }
}

// -----+
// | void CreateTable() |
// -----+
void CreateTable(int L){
    int Dim = 1;
    for(int pow=1 ; pow != L+1 ; ++pow) Dim *= 3 ;
    // Haplotype Table -----+;
    Vec Seq ;
    Seq.clear(); f(L+1, Seq);
    THap.erase(THap.begin() + THap.size());
    // Creation of Ta -----+;
    for(si=THap.begin() ; si!= THap.end() ; ++ si){
        int ak = 0;
        for(sj = *(si).begin(); sj!=*(si).end(); ++sj){
            if(*sj != 9) ak += 1 ;
        }
        Ta.push_back(ak);
    }
    // Mutation Table -----+;
    for(si=THap.begin() ; si!= THap.end() ; ++si){
        Seq.clear();
        for(int m=0 ; m!= L+1 ; ++m ){

```

```

    if ((*si).begin()+m ) != 1 ) Seq.push_back(-1);
    else{
        Vec Seqtmp;
        for(int im2 = 0 ; im2 <= L+1 ; ++im2){
            if( im2 == m ) Seqtmp.push_back(0);
            else Seqtmp.push_back( *((*si).begin()+im2) );
        }
        int Tk = NumSeq(Seqtmp);
        Seq.push_back(Tk);
    }
}
TMu.push_back(Seq);
}
// Ancestral Table -----;
for(si=THap.begin(); si!=THap.end(); ++si){
    Vec AncSeqTmp1 = f_ancestral(L , *si);
    TAnc.push_back(AncSeqTmp1);
}
// Recombination Table -----;
Vec SeqR , SeqL ;
int Is = -1;
for(si=THap.begin() ; si!= THap.end() ; ++si){
    SeqR.clear();
    SeqL.clear(); ++ Is ;
    int MinR = TAnc[Is][0];
    int MaxR = TAnc[Is][1];
    MinR = (int)*((*(TAnc.begin()+Is)).begin()+0);
    MaxR = (int)*((*(TAnc.begin()+Is)).begin()+1);

    Vec SeqTmp1 , SeqTmp2;
    for(int i=1 ; i!= MinR ; ++i) SeqR.push_back(-1);
    for(int i=1 ; i!= MinR ; ++i) SeqL.push_back(-1);
    for(int spot = MinR ; spot < MaxR ; ++spot){
        SeqTmp1.clear(); SeqTmp2.clear();
        for(int IM2 = 1 ; IM2 <= L+1 ; ++IM2){
            if(IM2 <= spot )
                SeqTmp1.push_back( *((*si).begin() + IM2 - 1) );
            else SeqTmp1.push_back(9);
            if(IM2 > spot )
                SeqTmp2.push_back( *((*si).begin() + IM2 - 1) );
            else SeqTmp2.push_back(9);
        }
        int Tj = NumSeq(SeqTmp1) ;
        SeqL.push_back(Tj);
        int Tk = NumSeq(SeqTmp2);
        SeqR.push_back(Tk);
    }
    for( int i=L+1 ; i!= MaxR ; --i) SeqL.push_back(-1);
    for( int i=L+1 ; i!= MaxR ; --i) SeqR.push_back(-1);
    TReL.push_back(SeqL);
    TReR.push_back(SeqR);
}
}

```

```

// Coalescence Table -----;
for(int Is1 = 0 ; Is1 != (int) THap.size(); ++Is1){
    Seq.clear();
    for(int Is2=0 ; Is2 != Is1 ; ++Is2){
        short int Possible = 1;
        short int Tk = -1 ;
        short int IM = 0;
        while(IM <= L ){
            if( THap[Is1][IM] == 0 && THap[Is2][IM] == 1 ) Possible=0;
            if( THap[Is1][IM] == 1 && THap[Is2][IM] == 0 ) Possible=0;
            if(Possible == 0) break ;
            ++IM;
        }
        if(Possible == 1 ){
            ListD EventC;
            // Creation of the new sequence k.
            Vec SeqTmp;
            for(IM=1 ; IM<=L+1 ; ++IM){
                if( THap[Is1][IM-1] == 0 || THap[Is2][IM-1] == 0 )
SeqTmp.push_back(0);
                if( THap[Is1][IM-1] == 1 || THap[Is2][IM-1] == 1 )
SeqTmp.push_back(1);
                if( THap[Is1][IM-1] == 9 && THap[Is2][IM-1] == 9 )
SeqTmp.push_back(9);
            }
            // Sequence already exist ?
            Tk = NumSeq(SeqTmp);
        }
        Seq.push_back(Tk);
    }
    TCoa.push_back(Seq);
}

double roundToS(double value , int F){
    double valuec = value * F ;
    valuec = arondi(valuec);
    valuec /= F ;
    return(valuec);
}

void roundS(double value , double& valuec , int& F ){
    F = 1 ;
    if(value < 1){
        while( int(value) == 0){
            value *= 10;
            F *= 10;
        }
        valuec = arondi(value);
    }
    else{

```

```

    valuec = arondi(value);
}
}

// +-----+
// | void OutPutTeX() |
// +-----+
void OutputTeX(Vec VNbCan , ListD Dist, long double elapsed_time ,
               int IndexMax , Time EndTime , Time StartTime ){
    string FileTEX = FileR + ".tex";
    const char* tex_file = FileTEX.c_str() ;
    ofstream out(tex_file, ios::out);
    if(!out){
        char *tex_back="MapArgDefault.tex";
        cerr << " + TeX file could not be opened for writing.\n";
        cerr << " + TeX File is then: MapArgDefault.tex.\n";
        ofstream out(tex_back, ios::out);
    }

    char *prefix = "";
    float K = NbSim;
    if(K>=1000000){
        K /= 1000000;
        prefix = "M";
    }
    else if(K>=1000){
        K /= 1000;
        prefix = "m";
    }
    float K2 = MeanNbSim;
    if(Phase == 1){
        char *prefix = "";
        if(K2>=1000000){
            K2 /= 1000000;
            prefix = "M";
        }
        else if(K2>=1000){
            K2 /= 1000;
            prefix = "m";
        }
    }
    string TimeB = StartTime.ascii() ;
    TimeB.resize( TimeB.length()-1 );
    string TimeE = EndTime.ascii() ;
    TimeE.resize( TimeE.length()-1 );
    int NbIntX = 8 , NbIntY = 8;

    out << "\\newcount\\landscape \\landscape=0"<<endl;
    out << "\\newcount\\LikBetweenInterval \\LikBetweenInterval=1 %(0,1)"<<endl;
    out << "\\newcount\\ConfidenceInterval \\ConfidenceInterval=1 %(0,1)"<<endl;
    out << "\\newcount\\LineAtTim \\LineAtTim=0 %(0,1)"<<endl;
    out << "\\newcount\\MarkerSeparation \\MarkerSeparation=1 % ( 0,1,2)"<<endl;

```

```

out << "\\newcount\\TicsLines \\TicsLines=0 %(0,1,2)"<<endl;
out << ""<<endl;
out << ""<<endl;
out << "\\ifnum\\the\\landscape=1 \\hsize=8.9true in \\vsize=6.5true in \\else
\\fi"<<endl;
out << "\\font\\sc=cmcsc10"<<endl;
out << "\\overfullrule=0pt"<<endl;
out << "\\headline{\\bf\\sc MapArg}\\copyright\\ (\\tt "<<program_name<<" / "
<<version_programm<<")\\hfill{\\tt "<<FileR<<"}" <<endl;
out << "\\bigskip"<<endl;
out << "\\footline{Report {\\tt [" << par_file [0] << par_file[1] << par_file[2]
<< K << prefix << RandomSeed <<"]} \\hfill "<<TimeB<<" /
"<<TimeE<<"}\\n"<<endl;
out << "\\def\\noir{\\special{color rgb 0 0 0}} ";
out << "\\def\\rouge{\\special{color rgb 0.92 0.71 0.42}} ";
out << "\\def\\spot{\\special{color rgb 0.83 0.05 0.18}} ";
out << "\\def\\vert{\\special{color rgb 0.61 0.93 0.42}} ";
out << "\\def\\gris{\\special{color rgb .74 .74 .87}} \\noir"<<endl;
out << "\\def\\lightgrey{\\special{color rgb .80 .80 .80}} \\noir"<<endl;
out << "\\long\\def\\boxit#1#2{\\vbox{\\hrule\\hbox{\\vrule\\kern#1"<<endl;
out << " \\vbox{\\kern#1\\vbox{#2}\\kern#1}\\vrule}\\hrule}"<<endl;
out << "\\centerline{\\bf\\boxit{.5em} {\\hfil Parameters\\hfil}
}\\n\\medskip"<<endl;
out << "\\medskip"<<endl;
out << "\\centerline{\\vbox{\\halign{\\hfil # & \\hskip.3cm {\\tt #}\\hfil
\\hskip0.6cm&"
<< "\\hfil# & \\hskip.3cm {\\tt #}\\hfil \\hskip0.6cm&"
<< "\\hfil# & \\hskip.3cm {\\tt #}\\hfil \\cr"<<endl;
out << "Data File & "<<dat_file<<" & $n$ & "<<N0<<" & $d$
&"<<NT<<"\\cr"<<endl;
if(Phase != 1) out << "$L-1$ &"<<NM<<" & $K$ & "<<K<<prefix<<" & $\\psi$ &
"<< MaxIter
<<"\\cr"<<endl;
if(Phase == 1) out << "$L-1$ &"<<NM<<" & $K$ & "<<K2<<prefix<<" & $\\psi$ &
"<< MaxIter
<<"\\cr"<<endl;
out << "Par. File & "<<par_file<<" & $r$ & "<<r<<" & $N_e$ & "<<Ne<<"
\\cr"<<endl;
out << "$\\rho$ & "<<rho<<" & $\\theta$ & "<<theta<<" & $\\kappa$ &
"<<kappa<<" \\cr"<<endl;
out << "$\\mu_1$ & "<<theta/(4*Ne*L)<<" & Simple & "<<Simple<<" & Re Type &
"<<ReType
<<" \\cr"<<endl;
out << "Seed & "<<RandomSeed<<" & Can & [" ;
for(unsigned int i=0 ; i != VNbCan.size() ; ++ i){
out << VNbCan[i] ;
if(i!=VNBcan.size()-1) out << " ; ";
}
out << "] & Res. phase & "<<Phase<<" \\cr"<<endl;
if(Phase != -1){
out << "Nb Graph & "<<NbGraphToKeep<<" & Rnd File & "<<rnd_file<<" &
$n_s$ & "

```

```

    << NResamp << " \cr"<<endl;
}
out << "}}}"<<endl;
out << "\\bigskip"<<endl;
out << "\\centerline{\\bf\\boxit{.5em} {\\hfil Results\\hfil} }\\n\\medskip"<<endl;
out << "\\medskip"<<endl;
out << "\\centerline{\\vbox{\\halign{\\hfil # & \\hskip.3cm {\\tt #}\\hfil
\\hskip0.6cm&}
    << "\\hfil# & \\hskip.3cm {\\tt #}\\hfil \\hskip0.6cm&}
    << "\\hfil# & \\hskip.3cm {\\tt #}\\hfil \\cr"<<endl;
out << " $\\hat{r}_T$ & "<<rt_estimate<<" & Interval & "<<IntervalMax<<" ";
out << "& Time/1M & "<< Mjourns<<"j "<<Mheures<<"h "<<Mminutes<<"m
"<<Msecondes
    <<"s \\cr"<<endl;
out << "\\% DropOut & [";
out.precision(4);
for(unsigned int i=0 ; i != VNbCan.size() ; ++ i){
    out << (Abandon[i]/NbSim*100) ;
    if(i != VNbCan.size()-1) out << ";";
}
out.precision(5);
out << "]" & CPU time& "<<CpuTime<<" & Real time & "<<RealTime<<"\\cr}}
"<<endl;
out << ""<<endl;
out << ""<<endl;

// Construction of the graph ;

// Min, Max and range ;
int IMinY = f_MinD(VY) ; double MinY = VY[IMinY] ;
int IMaxY = f_MaxD(VY) ; double MaxY = VY[IMaxY] ;
int IMinX = 0 ; double MinX = VX[IMinX] ;
int IMaxX = VX.size()-1 ; double MaxX = VX[IMaxX] ;
double EtX = MaxX - MinX ;
double EtY = MaxY - MinY ;
// Unit and Scale ;
double UnitX = EtX / NbIntX ;
double UnitY = EtY / NbIntY ;
double UnitX2 ; int FX ;
roundS(UnitX, UnitX2, FX) ;
double UnitY2 ; int FY ;
roundS(UnitY, UnitY2, FY) ;

double AMinY = -1 , AMaxY = -1 , AMinX = -1 , AMaxX = -1 ;
MinY = roundToS(MinY, FY) ;
MaxY = roundToS(MaxY, FY) ;
MinY = MinY*FY ; MaxY = MaxY*FY ;
while(AMinY == -1){
    MinY = arondi(MinY) - 1 ;
    if( int(MinY)%int(UnitY2) == 0) AMinY = MinY ;
}
while(AMaxY == -1){

```



```

    MaxY = arondi(MaxY) + 1;
    if( int(MaxY)%int(UnitY2) == 0) AMaxY = MaxY;
}
AMaxY /= FY ; AMinY /= FY ;
MinX = roundToS(MinX,FX) ;
MaxX = roundToS(MaxX,FX) ;
MinX *= FX; MaxX *= FX;
while(AMinX == -1){
    MinX = arondi(MinX) - 1;
    if( int(MinX)%int(UnitX2) == 0) AMinX = int(MinX);
}
while(AMaxX == -1){
    MaxX = arondi(MaxX) + 1;
    if( int(MaxX)%int(UnitX2) == 0) AMaxX = int(MaxX);
}
AMaxX /= FX ; AMinX /= FX ;
if(AMinX < 0) AMinX = 0;
EtX = AMaxX - AMinX;
EtY = AMaxY - AMinY;
// Rescale to make graph;
VecD VYR , VXR;
for(sjr=VY.begin(); sjr != VY.end(); ++sjr)
    VYR.push_back( 100*( *sjr - AMinY)/(AMaxY-AMinY) ) ;
for(sjr=VX.begin(); sjr != VX.end(); ++sjr)
    VXR.push_back( 100*( *sjr - AMinX)/(AMaxX-AMinX) ) ;

// Confidences intervals ;
double y0 = VY[IMaxY]-2 , b , a , x0;
VecD CI; Vec SL ;
double y_ic = VY[IMaxY]-2;
for(unsigned int itmp=0 ; itmp != VY.size()-1 ; ++ itmp){
    a = (VY[itmp+1] - VY[itmp]) / (VX[itmp+1] - VX[itmp]);
    b = VY[itmp] - a * VX[itmp];
    x0 = ( y0 - b) / a ;
    if(VY[itmp] < VY[itmp+1]){
        if(x0 >= VX[itmp] && x0 <= VX[itmp+1]){
            CI.push_back(x0);
            SL.push_back(1);
        }
    }
    if(VY[itmp] > VY[itmp+1]){
        if(x0 >= VX[itmp] && x0 <= VX[itmp+1]){
            CI.push_back(x0);
            SL.push_back(0);
        }
    }
}
out <<"\centerline{Approximate 95\% CI ";
for(unsigned int i=0 ; i != CI.size(); ++i){
if(i==0 && SL[i] == 0) out <<"[";
    if(SL[i] ==1) out << "[" << CI[i] ;
    if(SL[i] ==0) out << "]" << CI[i] << "]" ;
}

```

```

if(i==CI.size()-1 && SL[i] == 1) out <<"-]";
}
out << "}"<<endl;
out <<"\centerline{ESS [";
for(unsigned int i=0 ; i != ESS.size(); ++i){
    out << ESS[i] ;
    if(i != ESS.size()-1) out << " ; " ;
}
out << "]"<<endl;

// Descriptive Statistics ;
out <<"\centerline{Nb of graphs really done [";
int Size = VSNbC.size();
for(int i=0 ; i != Size; ++i){
    out << setprecision(4) << VNbGraphDone[i] ;
    if(i != Size-1) out << " ; " ;
} out << "]"<<endl;
out <<"\centerline{Nb of events [";
for(int i=0 ; i != Size; ++i){
    out << setprecision(4) << VSNbE[i] << "$\pm$" << setprecision(4) <<
sqrt(VS2NbE[i]) ;
    if(i != Size-1) out << " ; " ;
} out << "]"<<endl;
out <<"\centerline{Nb of Co. [";
for(int i=0 ; i != Size; ++i){
    out << setprecision(4) << VSNbC[i] << "$\pm$" << setprecision(4) <<
sqrt(VS2NbC[i]) ;
    if(i != Size-1) out << " ; " ;
} out << "]"<<endl;
out <<"\centerline{Nb of Mu. [";
for(int i=0 ; i != Size; ++i){
    out << setprecision(4) << VSNbM[i] << "$\pm$" << setprecision(4) <<
sqrt(VS2NbM[i]) ;
    if(i != Size-1) out << " ; " ;
} out << "]"<<endl;
out <<"\centerline{Nb of Re. [";
for(int i=0 ; i != Size; ++i){
    out << setprecision(4) << VSNbR[i] << "$\pm$" << setprecision(4) <<
sqrt(VS2NbR[i]) ;
    if(i != Size-1) out << " ; " ;
} out << "]"<<endl;

out << "\vfill"<<endl;
out << ""<<endl;
out << "% If you want to add points, lines, texts or others things on the
graphics,"<<endl;
    out << "% Note that the lower left corner of the main square is at (0,0), and the
upper"<<endl;
    out << "% right corner is at (100,100). Also, to obtain precise location for the X or Y
axis,"<<endl;

```

```

    out << "% use the following rule, where Xs is the X is the new coordinate
system:"<<endl;
    out << "%    Xs = (X-"<< AMinX<<")/"<<AMaxX-AMinX<<". "<<endl;
    out << "%    Ys = [Y-"<< AMinY<<"]/"<<AMaxY-AMinY<<". "<<endl;
    out << ""<<endl;
    out << ""<<endl;

    out << "\\input DraTeX.sty \\input AlDraTeX.sty "<<endl;
    out << "\\Define\\LineChart{\\NextTable{\\data(0,0)";
    out << "{\\MoveTo} \\data(0,999){\\LineTo}}\\Table\\data}"<<endl;
    out << "\\Draw"<<endl;
    out << " \\Ragged(1)"<<endl;
    out << " \\ifnum\\the\\landscape=1 \\Scale(5.5,4.5) \\else \\Scale(3,3) \\fi
"<<endl;
    out << " \\MoveTo(50,-10)\\Text(--$r_T$--)"
\\MoveTo(-15,50)\\Text(--$\\ln(L)$--)"<<endl;
    if(y_ic > AMinY)
    out << " \\ifnum\\the\\ConfidenceInterval=1\\vert\\PenSize(0.8pt)\\MoveTo(0,"
<<100*((VY[IMaxY]-2)-AMinY)/(AMaxY-AMinY)
<<")\\DoLine(100,0)(3){\\MoveF(-2)\\LineF(2)}\\noir\\else\\fi"<<endl;
    out << " \\MoveTo(0,0) \\MarkLoc(a)"<<endl;
    out << " \\MoveTo(100,100) \\MarkLoc(b)"<<endl;
    out << " \\DrawRectAt(0,0,100,100)"<<endl;
    out << " \\LineChart{";
VecD VCI ; // information of the delimitations between interval ;
for(unsigned int itmp=0 ; itmp != VY.size() ; ++ itmp){
    out << " "<<VXR[itmp]<<","<<VYR[itmp];
    if(itmp < VY.size()-1){
        if(VI[itmp] == VI[itmp+1] && itmp != VY.size()-1) out << " & ";
        if(VI[itmp] != VI[itmp+1]){
            VCI.push_back((VX[itmp]+VX[itmp+1])/2);
            out << "}"<<endl;
        }
    }
}
out << "}"<<endl;
for(unsigned int i=0 ; i != VY.size() ; ++ i){
    if(i < VY.size()-1){
        if(VI[i] != VI[i+1]){
            out << "\\ifnum\\the\\LikBetweenInterval=1
\\MoveTo("<<VXR[i]<<","<<VYR[i]<<")"
<< "\\LineTo("<<VXR[i+1]<<","<<VYR[i+1]<<") \\else \\fi"<<endl;
        }
    }
}

    out << "\\ifnum\\the\\MarkerSeparation=1{\\gris\\PenSize(0.8pt)";
for(unsigned int itmp=0 ; itmp != VCI.size() ; ++ itmp){
    out << "
\\gris\\PenSize(1pt)\\MoveTo("<<100*(VCI[itmp]-AMinX)/(AMaxX-AMinX)
<<","0)\\DoLine(0,100)(3){\\MoveF(-2)\\LineF(2)}\\noir"<<endl ;
}

```

```

    out << " \\noir}\\else\\fi"<<endl;
    out << "\\ifnum\\the\\MarkerSeparation=2{\\gris\\PenSize(0.8pt)";
    for(unsigned int itmp=0 ; itmp != VCI.size() ; ++ itmp){
    out <<"
\\gris\\PenSize(1pt)\\LineAt("<<100*( VCI[itmp]-AMinX)/(AMaxX-AMinX)<<"0,"
    << 100*( VCI[itmp]-AMinX)/(AMaxX-AMinX)<<"100)"<<endl;
    }
    out << " \\noir}\\else\\fi"<<endl;
    // Axes ;
    out <<" \\Axis(a,b)(W-1,";
    double ScaleY = AMinY , ScaleX = AMinX;
    while(ScaleY <= AMaxY + UnitY2/(2*FY)){
        out <<ScaleY;
    ScaleY += UnitY2/FY ;
        if(ScaleY <= AMaxY + UnitY2/(2*FY)) out <<"&";
    }
    out <<"")<<endl;
    out <<" \\Axis(a,b)(S-1,";
    while(ScaleX <= AMaxX + UnitX2/(2*FX)){
        out <<ScaleX;
    ScaleX += UnitX2/FX ;
        if(ScaleX <= AMaxX + UnitX2/(2*FX)) out <<"&";
    }
    out <<"")<<endl;
    out <<"{\\spot\\MoveTo("<<VXR[IMaxY]<<"1.5) \\Text(--$\\bigtriangleup$--)"
\\noir}"<<endl;
    out <<" \\ifnum\\the\\LineAtTim=1 \\spot\\LineAt("<< VXR[IMaxY]<<"0,"
    << VXR[IMaxY]<<"100)\\noir" <<"\\else\\fi"<<endl;

    out <<"{\\ifnum\\the\\ConfidenceInterval=1 \\vert\\PenSize(0.8pt)"<<endl;
    for(unsigned int i=0 ; i != CI.size() ; ++i){
        out <<"\\LineAt("<<100*( CI[i]-AMinX)/(AMaxX-AMinX)<<"0,"
        <<100*( CI[i]-AMinX)/(AMaxX-AMinX)<<"100,"
        <<100*(( VY[IMaxY]-2)-AMinY)/(AMaxY-AMinY)<<"n";
    }
    out << " \\noir\\else\\fi}"<<endl;

    // Tics Lines ;
    out <<" \\ifnum\\the\\TicsLines>0\\lightgrey\\PenSize(0.8pt)";
    ScaleY = AMinY + UnitY2/FY ; ScaleX = AMinX + UnitX2/FX;
    while(ScaleY <= AMaxY + UnitY2/(2*FY)){
        if(ScaleY < AMaxY) out << "\\ifnum\\the\\TicsLines=1\\LineAt(0,"
        <<100*(ScaleY-AMinY)/(AMaxY-AMinY)<<"100,"
        <<100*(ScaleY-AMinY)/(AMaxY-AMinY)<<"100)"\\else";
        if(ScaleY < AMaxY) out <<
"\\MoveTo(0,"<<100*(ScaleY-AMinY)/(AMaxY-AMinY)<<"100)"
        <<"\\DoLine(100,0)(2){\\MoveF(-1)\\LineF(1)}\\fi"<<endl ;
    ScaleY += UnitY2/FY ;
    }
    out << endl;
    while(ScaleX <= AMaxX + UnitX2/(2*FX)){
        if(ScaleX < AMaxX) out << "\\ifnum\\the\\TicsLines=1\\LineAt("

```

```

        <<100*(ScaleX-AMinX)/(AMaxX-AMinX)<<"0,"
        <<100*(ScaleX-AMinX)/(AMaxX-AMinX)<<"100)\else";
        if(ScaleX < AMaxX) out <<
"\\MoveTo("<<100*(ScaleX-AMinX)/(AMaxX-AMinX)<<"0)"
        <<"\\DoLine(0,100)(2){\\MoveF(-1)\\LineF(1)}\\fi"<<endl ;
        ScaleX += UnitX2/FX ;
    }
    out <<"\\else\\fi \\noir "<<endl;

    out << "\\EndDraw"<<endl;
    out << "\\eject\\bye"<<endl;
    out.close();
}

// +-----+
// | void WriteResult() |
// +-----+
void WriteResult(ListOfList Y0 , Vec VNbCan , ListD Dist, long double
elapsed_time ,
                int IndexMax , Time EndTime , Time StartTime ){

    string FileTXT = FileR + ".txt";
    const char* txt_file = FileTXT.c_str() ;
    ofstream out(txt_file,ios::out);
    if(!out){
        char *txt_back = "MapArgDefault.txt";
        cerr << " + Result File could not be opened for writing.\n";
        cerr << " + Result File is then: MapArgDefault.txt.\n";
        ofstream out(txt_back,ios::out);
    }

    out << " HapScan ["<<program_name<<"] (v. "<<version_programm <<")\n";
    out << "+ Data File:" << dat_file <<"\n";
    out << "+-----+\n";
    out << " NT = "<< NT << " / NM = " << NM << " / L = " << L << " / Sample
Size: "
        << N0 << "\n";
    out << " Sequence ...";
    for(int itemp1=0 ; itemp1 != NT; ++itemp1){
        out.width(2);
        out << " " << itemp1;
    }
    out << "\n";
    out << " Multiplicite ";
    for(i=Y0.begin() ; i != Y0.end(); ++ i){
        out.width(2);
        out << *(*i).begin() <<" " ;
    }
    out << "\n";
    out << " Trait ..... ";
    for(j=Trait.begin() ; j != Trait.end(); ++ j){
        out.width(2);

```

```

        if(*j == 9) out << " - ";
        else out << *j << " ";
    }
    out << "\n";
    for(int itemp2=1 ; itemp2 != NM+1 ; ++ itemp2){
        out << " M" << itemp2<<" ..... ";
        for(i=Y0.begin() ; i != Y0.end(); ++ i){
            out.width(2);
            if((*i).begin() + itemp2) != 9)
                out << *((*i).begin() + itemp2) << " ";
            else out << " - ";
        }
        out << "\n";
    }
    out << "\n";

    out << "+ Parameter File:" << par_file << "\n" ;
    out << "+-----+\n";
    out << "Sample Size:" << N0 << " for " << NT << " different sequences\n";
    out << "r=" << r << " / rho=" << rho << " / theta=" << theta << " / mu=" <<
theta/(4*Ne)
<< " / mu_1="<<theta/(4*Ne*L)<<endl;
    out << "kappa="<<kappa<<"\n";
    if(Phase == 1){
        NbSim = 0;
        for(unsigned int itemp11=1 ; itemp11 != VNbSim.size()+1 ; ++ itemp11)
            if( VNbCan[itemp11-1] > 0) NbSim += VNbSim[itemp11-1];
    }
    if(Phase != 1) out << "Likelihood on " << NbSim << " simulations. ReWeight = "
        << ReWeight << "\n";
    if(Phase == 1) out << "Likelihood on " << MeanNbSim << " simulations. ReWeight =
"
        << ReWeight << "\n";
    out << "[MaxIter " << MaxIter<<" ]\n";
    out << "[Recurrent Mutation " << RM <<" ]";
    out << "[Simple " << Simple<<" ]\n";
    out << "RandomSeed: " << RandomSeed << "\n";
    out << "Model for recombination: ";
    if(ReType == 0) out << "Original.\n";
    if(ReType == 1) out << "Random.\n";
    #ifdef Extnum
    out << "[Min(L)=" << MinLik << " / Max(L)=" << MaxLik << "]\n";
    #endif
    #ifndef Extnum
    out << "[Min(L)=" << MinLik / lambda << " / Max(L)=" << MaxLik / lambda <<
"]\n";
    #endif
    if(TraceRt == 1 ){
        out <<"[ TraceRt ="<<TraceRt<<" / FreqTraceRt = "<< FreqTraceRt<<"<<endl;
    }
    if(Phase == 1){
        out<<"[Phase: 1] ";
    }

```

```

out << "[Rnd File:" << rnd_file <<"] ";
out<<"[NS:"<<NResamp<<"]\n ";
out << "Input graphs [ ";
for(unsigned int itemp11=1 ; itemp11 != VG.size()+1 ; ++ itemp11){
    out << VG[itemp11-1] << " ";
}
out << "] NbSim [ ";
for(unsigned int itemp11=1 ; itemp11 != VNbSim.size()+1 ; ++ itemp11){
    if( VNbCan[itemp11-1] > 0)
        out << VNbSim[itemp11-1] << " ";
    else out <<"0 ";
}
out << "] \n";
}
if(Phase == 0){
    out<<"[Phase: 0] [Rnd File:" << rnd_file <<"] "<<"[NS:"<<NResamp
        <<"] [ G: "<<NbGraphToKeep<<"]\n ";
}
out << "Candidates [ ";
for(unsigned int itemp11=1 ; itemp11 != VNbCan.size()+1 ; ++ itemp11){
    out << VNbCan[itemp11-1] << " ";
}
out << "] \n";
out << "Driving Value [ ";
for(unsigned int itemp11=1 ; itemp11 != Driving.size()+1 ; ++ itemp11){
    if(Driving[itemp11-1] == -1) out << ". ";
else out << Driving[itemp11-1] << " ";
}
out << "] \n";
out << "DropOut [ ";
for(unsigned int itemp11=1 ; itemp11 != VNbCan.size()+1 ; ++ itemp11)
    out << (Abandon[itemp11-1]/NbSim) << " ";
out << "] \n";

// CPU Time ;
out << "CPU Time process: " << CpuTime << ".\n";

// Human Time ;
out << "Human Time process: " << RealTime << ".\n";

EXTNUM Time0 = EndTime.delta(&StartTime) ;
Time0 *= 1000000 ; Time0 /= NbSim ;
unsigned long int TimeProcess = int(Time0) ;
Mjours = (int)(long double)(TimeProcess / 86400) ;
TimeProcess -= Mjours * 86400 ;
Mheures = (int)(long double)(TimeProcess / 3600) ;
TimeProcess -= Mheures * 3600 ;
Mminutes = (int)(long double)(TimeProcess / 60) ;
TimeProcess -= Mminutes * 60 ;
Msecondes = TimeProcess ;

```

```

out.width(11);
out << "\n";
out << "+ Results: \n";
out << "+-----+\n";
out << "Can Int  Coordinate Lik_R           Lik           Ln(L)\n";
int CCand = 0 ; IntervalMax = -1 ;
long double PositionMax = -1 ;
double rglobal = 0 ;
for(unsigned int itemp3 = 1 ; itemp3 != Dist.size() + 1 ; ++ itemp3){
    double runit = Dist[itemp3-1] / (VNbCan[itemp3-1] + 1);
    for(int itmp2 = 1; itmp2 != VNbCan[itemp3-1] + 1; ++ itmp2){
        rglobal += runit ;
        out.width(3);
        out << CCand+1 ;
        out.width(3);
        out << itemp3 ;
        out.width(14);
        out << rglobal ;
        out.width(14);
        out << LikR[CCand] ;
        out.width(14);
        #ifdef Extnum
        out << LikE[CCand] ;
        #endif
        #ifndef Extnum
        out << LikE[CCand] / lambda ;
        #endif
        out.width(14);
        #ifdef Extnum
        out << log(Like[CCand]) << "\n";
        #endif
        #ifndef Extnum
        out << log(Like[CCand]) - log(lambda) << "\n";
        #endif
        VX.push_back(EXTNUM(rglobal*100));
        #ifdef Extnum
        VY.push_back(log(Like[CCand]));
        #endif
        #ifndef Extnum
        VY.push_back(log(Like[CCand]) - log(lambda));
        #endif
        VI.push_back(itemp3);
        CCand += 1 ;
        if(IndexMax == CCand - 1 ){
            PositionMax = rglobal;
            IntervalMax = itemp3 ;
        }
    }
    rglobal += runit ;
}
out << "\n";

```



```

out << "+ Maximum Likelihood Estimate: \n";
out << "+-----+\n";
out << "Candidat "<<IndexMax+1<<" in interval " << IntervalMax << " at position "
    << PositionMax*100 << "cM\n";
rt_estimate = PositionMax * 100 ;

// Descriptives Statistics ;
out << "\n";
out << "+ Descriptives Statistics. \n";
out << "+-----+\n";
out << "ESS [ ";
for(unsigned int itemp11=1 ; itemp11 != ESS.size()+1 ; ++ itemp11)
    out << ESS[itemp11-1] << " "; out << "]\n";
out << "Nb Graph really done [ ";
for(unsigned int itemp11=1 ; itemp11 != Dist0.size()+1 ; ++ itemp11)
    out << VNbGraphDone[itemp11-1] << " "; out << "]\n";
out << "Coalescence Event: Mean [ ";
for(unsigned int itemp11=1 ; itemp11 != Dist0.size()+1 ; ++ itemp11)
    out << VSNbC[itemp11-1] << " "; out << "]\n";
out << "
        Std [ ";
for(unsigned int itemp11=1 ; itemp11 != Dist0.size()+1 ; ++ itemp11)
    out << sqrt(VS2NbC[itemp11-1]) << " "; out << "]\n";
out << "Recombinat. Event: Mean [ ";
for(unsigned int itemp11=1 ; itemp11 != Dist0.size()+1 ; ++ itemp11)
    out << VSNbR[itemp11-1] << " "; out << "]\n";
out << "
        Std [ ";
for(unsigned int itemp11=1 ; itemp11 != Dist0.size()+1 ; ++ itemp11)
    out << sqrt(VS2NbR[itemp11-1]) << " "; out << "]\n";
out << "Mutation Event: Mean [ ";
for(unsigned int itemp11=1 ; itemp11 != Dist0.size()+1 ; ++ itemp11)
    out << VSNbM[itemp11-1] << " "; out << "]\n";
out << "
        Std [ ";
for(unsigned int itemp11=1 ; itemp11 != Dist0.size()+1 ; ++ itemp11)
    out << sqrt(VS2NbM[itemp11-1]) << " "; out << "]\n";
out << "Number Events by graph: [ ";
for(unsigned int itemp11=1 ; itemp11 != Dist0.size()+1 ; ++ itemp11)
    out << VSNbE[itemp11-1] << " "; out << "]\n";
out << "
        Std [ ";
for(unsigned int itemp11=1 ; itemp11 != Dist0.size()+1 ; ++ itemp11)
    out << sqrt(VS2NbE[itemp11-1]) << " "; out << "]\n";

out.close();
}

// +-----+
// | void LireParm() |
// +-----+
void LireParm(){
string buf;

#ifdef DEBUG
if(debug > 1) cout <<"Reading Parameter File\n" ;

```

```

#endif
ifstream in(par_file);
if (!in) {
    cerr << "ERROR: Parameter file " <<par_file<<" could not be opened";
    exit(1);
}
while(getline(in, buf) ){
    string::const_iterator cp = buf.begin();
    int cnt = 0;
    int cas = 0;
    if( *cp == '#' continue;
    while(cp != buf.end()){
        while(*cp == ' ') cp ++;
        if(*cp ){
            cnt++;
            string mot;
            while(*cp != ' ' && cp != buf.end()){
                mot += *cp;
                cp++;
            }
            if(cnt == 1){
                if(mot == "Ne") cas = 1;
                if(mot == "mu") cas = 2;
                if(mot == "Dist") cas = 4;
                if(mot == "RecurrentMutation") cas = 5;
                if(mot == "NbSim") cas = 6;
                if(mot == "NbCan") cas = 7;
                if(mot == "VNbCan") cas = 8;
                if(mot == "NbGraphToKeep") cas = 9;
                if(mot == "ReWeight") cas = 10;
                if(mot == "RndFile") cas = 11;
                if(mot == "Phase") cas = 12;
                if(mot == "ResultFile") cas = 13;
                if(mot == "DataFile") cas = 14;
                if(mot == "Simple") cas = 15;
                if(mot == "kappa") cas = 16;
                if(mot == "PrintNbSim") cas = 17;
                if(mot == "MaxIter") cas = 18;
                if(mot == "FreqTraceRt") cas = 20;
                if(mot == "TraceRt") cas = 21;
                if(mot == "ReType") cas = 22;
                if(mot == "TeXOutput") cas = 24;
                if(mot == "Driving") cas = 27;
                if(mot == "NResampling") cas = 28;
                if(mot == "AncestralRW") cas = 29;
                if(cas == 0){
                    cerr << "Wrong parameter: " << mot << "." << endl;
                }
            }
        }
        if(cnt > 1){
            if(cas == 1) Ne = atoi( mot.c_str() );
            if(cas == 2) Mu0.push_back( atof( mot.c_str() ) );
        }
    }
}

```

```

        if(cas == 4) Dist0.push_back(atoi( mot.c_str() ));
        if(cas == 5) RM = bool( atoi( mot.c_str() ));
        if(cas == 6) NbSim = atoi( mot.c_str() );
        if(cas == 7) NbCan = atoi( mot.c_str() );
        if(cas == 8) VNbCan.push_back(atoi( mot.c_str() ));
        if(cas == 9) NbGraphToKeep = atoi( mot.c_str() );
        if(cas == 10) ReWeight = atof( mot.c_str() );
        if(cas == 11) rnd_file = ConvertSC2( mot );
        if(cas == 12) Phase = atoi( mot.c_str() );
        if(cas == 13) res_file = ConvertSC2( mot );
        if(cas == 14) dat_file = ConvertSC2( mot );
        if(cas == 15) Simple = atoi( mot.c_str() );
        if(cas == 16) kappa = atof( mot.c_str() );
        if(cas == 17) PrintNbSim = atoi( mot.c_str() );
        if(cas == 18) MaxIter = atoi( mot.c_str() );
        if(cas == 20) FreqTraceRt = atoi( mot.c_str() );
        if(cas == 21) TraceRt = atoi( mot.c_str() );
        if(cas == 22) ReType = atoi( mot.c_str() );
        if(cas == 24) TeXOutput = atoi( mot.c_str() );
        if(cas == 27) DrivingInitial.push_back( atof(mot.c_str() ));
        if(cas == 28) NResamp = atoi( mot.c_str() );
    }
}
}
}

// +-----+
// | void f_MaxProd() |
// +-----+
void f_MaxProd(VecE ProdFE){
    if(Phase == 0){
        long double rt = VCR[Max_pos];
        VTraceRt.push_back(rt);
    }
    else{
        int IndexMax = f_Max(ProdFE);
        long double rt = VCR[IndexMax];
        VTraceRt.push_back(rt);
    }
}

// +-----+
// | void f_MaxLik() |
// +-----+
void f_MaxLik(int simu){
    int IndexMax = f_Max(LikE);
    long double rt = VCR[IndexMax];
#ifdef Extnum
    long double rtl = double(log(LikE[IndexMax]/((double)(simu)))) ;
#endif
#ifdef Extnum

```

```

    long double rtl = double(log(LikE[IndexMax]/((double)(simu))) -
(double)(log(lambda))) ;
    rtl = log( (LikE[IndexMax]/lambda) / simu ) ;

#ifdefif
if( fabs(rt - last_rt_TraceRt) > 0.000001 || fabs(rtl - last_rtL_TraceRt)>0.01 ){
    VTraceRtI.push_back(IndexMax);
    VTraceRtL.push_back( rtl );
    VTraceRtS.push_back(simu);
    last_rt_TraceRt = rt ;
    last_rtL_TraceRt = rtl ;
}
}

// +-----+
// | LireRnd() |
// +-----+
void LireRnd(int interval){
    LikStep.clear();
    LikSeed.clear();
    LikVal.clear();
    LikM.clear();
    string buf;
    string FileRND = (string)(rnd_file) + ".rnd";
    const char* rnd_file = FileRND.c_str() ;
    ifstream in(rnd_file);
    int nb1 = 0 ; EXTNUM nb2 ;
    int nb3 ; unsigned long nb4 ;
    if (!in) {
        cerr << "ERROR: RND file " <<rnd_file<<" could not be opened";
        exit(1);
    }
    while(getline(in, buf)){
        string::const_iterator cp = buf.begin();
        int cnt = 0 , flag = 0; VecE LikT ;
        while(cp != buf.end()){
            while(*cp == ' ') cp ++;
            if(*cp ){
                cnt++;
                string mot;
                while(*cp != ' ' && cp != buf.end()){
                    mot += *cp;
                    cp++;
                }
                if(cnt ==1) nb1 = atoi(ConvertSC(mot)) ;
                if(cnt ==1 && mot == "L") flag = 1;
                if(cnt ==2 && flag == 0) nb2 = atofd(mot.c_str()) ;
                if(cnt ==3 && flag == 0) nb3 = atoi(ConvertSC(mot)) ;
                if(cnt ==4 && flag == 0) nb4 = strtoul(ConvertSC(mot), NULL, 10) ;
            }
            if(flag == 1){
                if(cnt > 1) LikM.push_back( atofd(mot.c_str())) ;
            }
        }
    }
}

```

```

    }
}

if(nb1 == interval){
    if(flag == 0) LikStep.push_back(nb3);
    if(flag == 0) LikSeed.push_back(nb4);
    if(flag == 0) LikVal.push_back(nb2);
}
}
if(NbGraphToKeep>0 && NbGraphToKeep < LikStep.size())
VG.push_back(NbGraphToKeep);
else VG.push_back(int(LikStep.size()));
if(NbGraphToKeep>0 && NbGraphToKeep < LikStep.size()){
    LikStep.assign(LikStep.begin(), LikStep.begin()+NbGraphToKeep);
    LikVal.assign(LikVal.begin(), LikVal.begin()+NbGraphToKeep);
    LikSeed.assign(LikSeed.begin(), LikSeed.begin()+NbGraphToKeep);
}
#ifdef Extnum
for(sje=LikM.begin(); sje != LikM.end() ; ++sje) *sje *= lambda ;
#endif
}

// +-----+
// | LisOfList LireData() |
// +-----+
ListOfList LireData(){
    ListOfList GListe;
    string buf;

#ifdef DEBUG
    if(debug > 1) cout <<"Reading Data File\n" ;
#endif
    ifstream in(dat_file);
    if (!in) {
        cerr << "ERROR: Data file " <<dat_file<<" could not be opened";
        exit(1);
    }
    while(getline(in, buf)){
        string::const_iterator cp = buf.begin();
        int cnt = 0;
        List list1;
        while(cp != buf.end()){
            while(*cp == ' ') cp ++;
            if(*cp){
                cnt++;
                string mot;
                while(*cp != ' ' && cp != buf.end()){
                    mot += *cp;
                    cp++;
                }
                if(cnt != 2){

```

```

        if(mot == "-") mot = "9";
        list1.push_back(atoi(ConvertSC(mot)));
        if(cnt == 1 ) N0 = N0 + (int)atoi(ConvertSC(mot));
    }
    if(cnt == 2){
        if(mot == "-") mot = "9";
        Trait.push_back(atoi(ConvertSC(mot)));
    }
}
}
NM = cnt - 2;
GListe.push_back(list1);
}
return(GListe);
}

// +-----+
// | void PossCoaId() |
// +-----+
void PossCoaId(short int Is){
    if( TNi[Is] > 1 ){
        VecSI TEvent;
        TEvent.push_back(1);
        TEvent.push_back(Is);
        PEvent.push_back(TEvent);
        Prob.push_back(0);
    }
}

// +-----+
// | void PossRecom() |
// +-----+
void PossRecom(short int Is ){
    VecSI TEvent;
    TEvent.push_back(4);
    TEvent.push_back(0);
    TEvent.push_back(0);
    if(Simple == 1){
        TEvent.push_back(0);
        TEvent.push_back(0);
    }
    for(int Spot=1 ; Spot != L ; ++Spot){
        if( (Spot >= TAnc[Is][0]) && (Spot < TAnc[Is][1]) ){
            TEvent[1] = Is;
            TEvent[2] = Spot;
            Prob.push_back(0);
            PEvent.push_back(TEvent);
        }
    }
}

// +-----+

```

```

// | void PossCoadi() |
// +-----+
void PossCoadi(short int ID1, short int ID2){
    int ID3 = -1;
    if(ID1 < ID2) ID3 = TCoa[ID2][ID1];
    if(ID1 > ID2) ID3 = TCoa[ID1][ID2];
    if(ID3 >= 0){
        VecSI TEvent;
        TEvent.push_back(2);
        TEvent.push_back(ID1);
        TEvent.push_back(ID2);
        TEvent.push_back(ID3);
        PEvent.push_back(TEvent);
        Prob.push_back(0);
    }
}

// +-----+
// | void PossMut() |
// +-----+
void PossMut(VecSI MuP){
    VecSI TEvent;
    for(sjk1=MuP.begin(); sjk1 != MuP.end(); ++sjk1){
        TEvent.clear();
        // Recherche de l'haplotype qui peut muter ;
        short int Ti = -1;
        sjk = ID.begin();
        while(Ti == -1){
            if(THap[*sjk][*sjk1] == 1) Ti = *sjk;
            ++sjk;
        }
        TEvent.push_back(3); // Mutation ;
        TEvent.push_back(*sjk1); // Mutate Base ;
        TEvent.push_back(Ti); // Sequence qui mute ;
        TEvent.push_back( TMu[Ti][*sjk1] ); // Type de la nouvelle sequence eventuelle ;
        PEvent.push_back(TEvent);
        Prob.push_back(0); // Prob ;
    }
}

// +-----+
// | void PossMutRM() |
// +-----+
void PossMutRM(short int Is){
    VecSI TEvent;
    for(short int l=0; l != L; ++l){
        if(THap[Is][l] != 1) continue;
        TEvent.clear();
        TEvent.push_back(3); // Mutation ;
        TEvent.push_back(l); // Mutate Base ;
        TEvent.push_back(Is); // Sequence qui mute ;
        TEvent.push_back( TMu[Is][l] ); // Type de la nouvelle sequence eventuelle ;
    }
}

```

```

    PEvent.push_back(TEvent);
    Prob.push_back(0); // Prob ;
}
}

// +-----+
// | void Usage() |
// +-----+
void usage(void){
    cerr << " Usage is " <<program_name << " -p Parameter_File [options]\n";
    cerr << " Options:\n";
    cerr << " -d x      debug set to level x\n";
    cerr << " -g x      request gnuplot output, x=(0,1)\n";
    cerr << " -r x      set random seed to x (x is integer)\n";
    cerr << " -e x      estimate time to do x iterations\n";
    cerr << " -o x      output file is \"x.txt\", \"x.plt\", \"x.don\",
\"x.tex\"... \n";
    cerr << " -K x      do x iterations\n";
    cerr << " -G x      Nb of Graph to keep for resampling\n";
    cerr << " -N x      Resampling at N=x \n";
    cerr << " -R x      restore a crashed process\n";
    cerr << " -Z rnd_file to set the rnd file\n";
    exit(8);
}

//+-----+;
//| TimeNextEvent() |;
//+-----+;
double TimeNextEvent(){
    converge = 1 ;
    double U = mtrand2.rand(1);
    double ta = -N*(N-1)/(2*kappa);
    double tb = kappa;
    double tc = -N*theta/2 - N*rho/2;
    double td = N*(N-1)*exp(kappa*c_time)/(2*kappa) + N*theta*c_time/2 +
N*rho*c_time/2 -log(U);
    double x = ta*tb*exp(-tb*td/tc)/tc;
    /* ANSI C code for W(x). K M Briggs 98 Feb 12
    http://www.btxact.com/people/briggsk2/W-ology.html
    Based on Halley iteration. Converges rapidly for all valid x. */
    int i;
    double p , e , t , w , eps = 4.0e-16; /* eps=desired precision */
    if (x<-0.36787944117144232159552377016146086) {
        fprintf(stderr,"x=%g is < -1/e, exiting.\n",x);
        exit(1);
    }
    if ( x == 0.0 ) return 0.0;
    /* get initial approximation for iteration... */
    if (x < 1.0) { /* series near 0 */
        p = sqrt(2.0*(2.7182818284590452353602874713526625*x+1.0));
        w = -1.0+p-p*p/3.0+11.0/72.0*p*p*p;
    }
}

```



```

else w = log(x); /* asymptotic */
if (x>3.0) w -= log(w);
for (i=0; i<20; i++) { /* Halley loop */
    e = exp(w);
    t = w*e-x;
    t /= e*(w+1.0)-0.5*(w+2.0)*t/(w+1.0);
    w -= t;
    //if (fabs(t)<eps*(1.0+fabs(w))) return w; /* rel-abs error */
    if (fabs(t)<eps*(1.0+fabs(w))) {
        return(-(w*tc + td*tb)/(tc*tb)) ; /* rel-abs error */ }
} /* never gets here */
fprintf(stderr,"No convergence at x=%g\n",x);
converge = 0 ;
return(0);
}

//+-----+;
//|      TimeNextEvent2()                                     |;
//+-----+;
double TimeNextEvent2(){
    converge = 1 ;
    double U1 = mtrand2.rand(1);
    double U2 = mtrand2.rand(1);
    double s1 = log(exp(kappa*c_time)
-2*kappa*log(U1)/((double)(N)*((double)(N)-1)))/kappa ;
    double s2 = -(2/(double)(N))*log(U2) + c_time*(alpha*theta
+beta*rho)/(alpha*theta + beta*rho);
    if(s2 < s1) return(s2);
    else return(s1);
}

// +-----+
// | void CalculProb()                                     |
// +-----+
void CalculProb(){
    for(short int Ev=0 ; Ev != PEvent.size() ; ++Ev){
        if (PEvent[Ev][0] == 1){
            int Ti = PEvent[Ev][1] ;
            if(VPS == 0) Prob[Ev] = N*( TNi[Ti] -1 );
            if(VPS == 1) Prob[Ev] = lambda_t*N*( TNi[Ti] -1 );
        }
        else if ( PEvent[Ev][0] == 2){
            int DeltaIK = 0, DeltaJK = 0 , nk ;
            if( PEvent[Ev][1] == PEvent[Ev][3] ) DeltaIK = 1 ;
            if( PEvent[Ev][2] == PEvent[Ev][3] ) DeltaJK = 1 ;
            nk = TNi[ PEvent[Ev][3] ];
            if(VPS == 0)
                Prob[Ev] = 2*N*(nk + 1 - DeltaIK - DeltaJK);
            if(VPS == 1)
                Prob[Ev] = 2*lambda_t*N*(nk + 1 - DeltaIK - DeltaJK);
        }
        else if( PEvent[Ev][0] == 3){

```

```

        int nj = TNi[ PEvent[Ev] [3] ];
        Prob[Ev] = theta*(nj + 1)/L;
    }
    else if( PEvent[Ev] [0] == 4){
        short int spot = PEvent[Ev] [2] ;
        if(ReType == 1){
            int np = VNprime[spot-1] ;
            Prob[Ev] = (ReWeight*rho*np*( *(Dist.begin() + spot -1) / r) ) /
(N+1);
        }
        if(ReType == 0){
            int Ti = PEvent[Ev] [1] ;
            int Tj = TReL[Ti] [spot-1];
            int Tk = TReR[Ti] [spot-1];
            int nj = TNi[Tj] , nk = TNi[Tk] ;
            if(Simple == 1){
                int njp = PEvent[Ev] [3];
                int nkp = PEvent[Ev] [4];
                Prob[Ev] = (ReWeight*rho*(njp+1)*(nkp+1)
                    *( *(Dist.begin() + spot -1) / r) ) / (N+1);
            }
            else Prob[Ev] = (ReWeight*rho*(nj+1)*(nk+1)
                *( *(Dist.begin() + spot -1) / r) ) / (N+1);
        }
    }
}
}

//+-----+;
//|   ChooseParents()                               |;
//+-----+;
void ChooseParents(){
    short int spot = PEvent[ChoosenEvent] [2];
    short int Ti = PEvent[ChoosenEvent] [1];
    Tj = SeqPosL(Ti);
    Tk = SeqPosR(Ti);
    if(Tj == -1) Tj = TReL[Ti] [spot-1];
    if(Tk == -1) Tk = TReR[Ti] [spot-1];
    if(debug>50) cout<<"[Ti "<<Ti<<"] [Tj "<<Tj<<"] Tk="<<Tk<<">>endl;
    n_j = TNi[Tj] ; n_k = TNi[Tk];
}

// +-----+
// | void ChooseEvent()                               |
// +-----+
void ChooseEvent(){
    SommeP = 0 ;
    for(sjr=Prob.begin(); sjr!=Prob.end(); ++sjr){
        SommeP += *sjr ;
    }
    long double Alea = mtrand2.rand(SommeP);
    ChoosenEvent = -1 ;
}

```

```

long double PMin = 0 , PMax = 0 ;
int index_choix = -1;
sjr = Prob.begin();
while(ChoosenEvent == -1){
    ++ index_choix ;
    PMin = PMax ;
    PMax += *sjr ;
    if(PMin != PMax && PMin <= Alea && Alea <= PMax) ChoosenEvent =
index_choix ;
    ++sjr;
}
TypeEv = PEvent[ChoosenEvent] [0];
}

// +-----+
// | void DeleteCoaId() |
// +-----+
void DeleteCoaId(int IdSeq){
    int EtoDel = - 1;
    int Index = 0;
    while(EtoDel == -1 ){
        if(PEvent[Index] [0] == 1 && PEvent[Index] [1] == IdSeq) EtoDel = Index ;
        ++ Index ;
    }
    PEvent.erase(PEvent.begin() + EtoDel);
    Prob.erase(Prob.begin() + EtoDel);
}

// +-----+
// | void AddNprime() |
// +-----+
void AddNprime(int NS){
    int begin_tmp = TAnc[NS] [0];
    int end_tmp = TAnc[NS] [1];
    for(int Interv=begin_tmp-1 ; Interv != end_tmp -1; ++Interv)
        VNprime[Interv] += 1 ;
}

// +-----+
// | void DeleteSandE() |
// +-----+
void DeleteSandE(int NS){
    // Remove events when sequence NS is involved in event ;
    List EventToDelete;
    for(int Index =0 ; Index != int(PEvent.size()) ; ++ Index ){
        if( PEvent[Index] [0] == 1){
            if( PEvent[Index] [1] == NS )
                EventToDelete.push_back(Index) ;
        }
        if( PEvent[Index] [0] == 2){
            if( PEvent[Index] [1] == NS || PEvent[Index] [2] == NS )
                EventToDelete.push_back(Index) ;
        }
    }
}

```

```

    }
    else if( PEvent[Index][0] == 3){
        if( PEvent[Index][2] == NS )
            EventToDelete.push_back(Index) ;
    }
    else if( (ReType == 0) && (PEvent[Index][0] == 4)){
        if( PEvent[Index][1] == NS )
            EventToDelete.push_back(Index) ;
    }
}

itmp = 0;
for(j=EventToDelete.begin() ; j !=EventToDelete.end() ; ++j){
    PEvent.erase(PEvent.begin()+ *j - itmp);
    Prob.erase(Prob.begin()+ *j - itmp);
    itmp ++ ;
}

// Adjust n' vector ;
if(ReType == 1){
    int begin_tmp = TAnc[NS][0];
    int end_tmp = TAnc[NS][1];
    for(int Interv=begin_tmp-1 ; Interv != end_tmp -1 ; ++Interv)
        VNprime[Interv] -= 1 ;
}

// Remove in ID vector ;
int position = -1 , index = 0 ; sjk = ID.begin();
while(position == -1){
    if( *sjk == NS) position = index ;
    ++ index; ++sjk;
}
ID.erase(ID.begin() + position);
}

// +-----+
// | void CheckRemoveMu() |
// +-----+
void CheckRemoveMu(int IdSeq){ // Delete event of CoaId for sequence IdSeq ;
    List LEtoDel ; // Number of the event to delete ;
    int IndexdEtoDel = 0;
    for(sik=PEvent.begin() ; sik != PEvent.end() ; ++sik){
        if(*((*sik).begin()) == 3 && *((*sik).begin()+2) == IdSeq)
            LEtoDel.push_back(IndexdEtoDel) ;
        ++ IndexdEtoDel ;
    }
    itmp = 0;
    for(j=LEtoDel.begin() ; j !=LEtoDel.end() ; ++j){
        PEvent.erase(PEvent.begin()+*j - itmp);
        Prob.erase(Prob.begin()+*j - itmp);
        itmp ++ ;
    }
}

```

```

// +-----+
// | void UpdateY() |
// +-----+
void UpdateY(){
    VecSI CheckMu ;
    VecSI Event = PEvent[ChoosenEvent] ;
    // +-----+
    // | Coalescence of two sequences of the same type |
    // +-----+
    if(TypeEv == 1) {
        int Ti , n_i ;
        Ti = Event[1] ;
        TNi[Ti] -= 1 ;
        n_i = TNi[Ti] ;
        N -= 1 ;
    }
    if(Simple == 1) del_seq_tree(Ti) ;
    if(RM == 0){
        for(itmp = 0 ; itmp <= L-1; ++itmp){
            if( THap[Ti][itmp] != 9){
                if(n_i == 1){
                    if( VMutation[itmp] == 2 && THap[Ti][itmp] == 1)
                        CheckMu.push_back( itmp ) ;
                }
                VMutation[itmp] -= THap[Ti][itmp] ;
            }
        }
        a -= Ta[Ti] ; b -= Tb[Ti] ;
        if(n_i == 1){
            PEvent.erase(PEvent.begin() + ChoosenEvent) ;
            Prob.erase(Prob.begin() + ChoosenEvent) ;
            if(CheckMu.size() > 0) PossMut(CheckMu) ;
        }
    }
    // +-----+
    // | Coalescence of two sequences of differents types i and j in sequence k |
    // +-----+
    else if(TypeEv == 2) {
        int Ti , Tj, Tk , n_i , n_j , n_k = -1 ;
        List CheckMui , CheckMuj , VMutation_O , VMutPoss ;
        if(RM == 0){
            for(itmp=0 ; itmp != L ; ++ itmp) VMutPoss.push_back(0) ;
            VMutation_O.assign(VMutation.begin() , VMutation.end()) ;
        }
        Ti = Event[1] ;
        Tj = Event[2] ;
        Tk = Event[3] ;
        TNi[Ti] -= 1 ;
        TNi[Tj] -= 1 ;
        n_i = TNi[Ti] ;
        n_j = TNi[Tj] ;
    }
}

```

```

N -= 1;
if(Simple == 1) del_seq_tree(Ti);
if(Simple == 1) del_seq_tree(Tj);
if(RM == 0){
  for(itmp = 0 ; itmp <= L-1; ++itmp){
    if( THap[Ti][itmp] != 9){
      if(n_i == 0){
        if( ( VMutation[itmp] == 1 && THap[Ti][itmp] == 1) && (Ti !=
Tk) )
          *(VMutPoss.begin()+itmp) = 1 ;
      }
      VMutation[itmp] -= THap[Ti][itmp] ;
    }
    if( THap[Tj][itmp] != 9){
      if(n_j == 0){
        if( ( VMutation[itmp] == 1 && THap[Tj][itmp] == 1) && (Tj !=
Tk) )
          *(VMutPoss.begin()+itmp) = 1 ;
      }
      VMutation[itmp] -= THap[Tj][itmp] ;
    }
  }
}
// Modification of a and b ;
a -= Ta[Ti] ; b -= Tb[Ti] ;
a -= Ta[Tj] ; b -= Tb[Tj] ;
// New sequence k
TNi[Tk] += 1;
n_k = TNi[Tk];
if(Simple == 1) add_seq_tree(Tk);
if(RM == 0){
  for(itmp = 0 ; itmp <= L-1; ++itmp){
    if( THap[Tk][itmp] != 9){
      if( VMutation[itmp] == 0 && THap[Tk][itmp] == 1 )
        CheckMu.push_back(itmp + 1 ) ;
      VMutation[itmp] += THap[Tk][itmp] ;
    }
  }
}
if((n_k == 1) && (Ti != Tk) && (Tj != Tk) && (ReType == 1))
AddNprime(Tk);
a += Ta[Tk] ; b += Tb[Tk] ;
// Events to modify ;
if(Ti == Tk){
  if(n_j == 0){
    DeleteSandE(Tj) ;
  }
  if(n_j == 1) DeleteCoaId(Tj) ;
}
if(Tj == Tk){
  if(n_i == 0){
    DeleteSandE(Ti) ;

```

```

    }
    if( $n_i == 1$ ) DeleteCoaId( $T_i$ ) ;
}
if( $T_i != T_k$  &&  $T_j != T_k$ ){
    if( $TN_i[T_k] == 1$ )  $ID.push\_back( T_k )$ ;
    if( $TN_i[T_k] == 1$ ){
        if( $ReType == 0$ ) PossRecom(  $T_k$  );
        if( $RM == 1$ ) PossMutRM (  $T_k$  );
    }
    if( $n_i == 0$  ){
        DeleteSandE( $T_i$ ) ;
    }
    if( $n_i == 1$ ) DeleteCoaId(  $T_i$  );
    if( $n_j == 0$ ){
        DeleteSandE( $T_j$ ) ;
    }
    if( $n_j == 1$ ) DeleteCoaId(  $T_j$  );
}
if( $n_k == 1$  &&  $T_i != T_k$  &&  $T_j != T_k$  ){
    for( $sjk1=ID.begin()$  ;  $sjk1 !=ID.end()$  ; ++ $sjk1$ ){
        if( $*sjk1 == T_k$ ) continue ;
        PossCoaDi( $*sjk1$  ,  $T_k$ );
    }
}
if( $n_k == 2$ ){
    if(( $T_i != T_k$ ) && ( $T_j != T_k$ )) PossCoaId(  $T_k$  ) ;
}
if( $RM == 0$ ){
    VecSI CheckMuT;
    for(int  $indexmu = 0$  ;  $indexmu < L$  ; ++ $indexmu$ ){
        if(  $VMutation\_O[indexmu] == 2$  &&  $VMutation[indexmu] == 1$  )
             $VMutPoss[indexmu] = 1$  ;
    }
    for(int  $indexmu = 0$  ;  $indexmu < L$  ; ++ $indexmu$ ){
        if(  $VMutPoss[indexmu] == 1$  )  $CheckMuT.push\_back(indexmu)$ ;;
    }
    if( $CheckMuT.size() > 0$ ) PossMut( $CheckMuT$ ); // Generate a new mutation event
;
}
}

// +-----+
// | Mutation of sequence i in sequence j |
// +-----+
else if( $TypeEv == 3$ ) {
    int  $T_i$  ,  $T_j$ ,  $n_j$  ;

     $T_i = Event[2]$  ;
     $T_j = Event[3]$  ;
    if( $Simple == 1$ ){
        del_seq_tree( $T_i$ ) ;
        add_seq_tree( $T_j$ ) ;
    }
}

```

```

}
TNi[Ti] -= 1;
// Update of VMutation ;
if(RM == 0){
    for(itmp = 0 ; itmp <= L-1; ++itmp){
        if( THap[Ti][itmp] != 9) VMutation[itmp] -= THap[Ti][itmp] ;
    }
}
// Modifications of a and b ;
a -= Ta[Ti] ; b -= Tb[Ti] ;
// New sequence j
if(TNi[Tj] == 0) ID.push_back( Tj ) ;
if(TNi[Tj] == 0){
    if(ReType == 0) PossRecom( Tj ) ;
    if(RM == 1) PossMutRM( Tj ) ;
}
TNi[Tj] += 1;
n_j = TNi[Tj];
if(RM == 0){
    for(itmp = 0 ; itmp <= L-1; ++itmp){
        if( THap[Tj][itmp] != 9){
            if( VMutation[itmp] == 0 && THap[Tj][itmp] == 1 )
                CheckMu.push_back(itmp) ;
            VMutation[itmp] += THap[Tj][itmp] ;
        }
    }
}
if(ReType == 1) if(n_j == 1) AddNprime(Tj);
a += Ta[Tj] ; b += Tb[Tj] ;

if(TNi[Ti] == 0) DeleteSandE(Ti); // Delete sequence i and associated events ;
if(n_j == 1){
    for(sjk1=ID.begin() ; sjk1 !=ID.end() ; ++sjk1){
        if(*sjk1 == Tj) continue ;
        PossCoaDi(*sjk1 , Tj);
    }
    if(CheckMu.size() > 0) PossMut(CheckMu);
}
if(n_j == 2){
    if(Ti != Tj) PossCoaId( Tj ) ;
}
}

// +-----+
// | Recombination of sequence i into sequences j and k |
// +-----+
else if( TypeEv == 4) {
    if(ReType == 0){
        if(Simple == 1) ChooseParents();
        int spot = Event[2] ;
        Ti = Event[1] ;
        if(Simple == 0){

```



```

        Tj = TReL[Ti][spot-1];
        Tk = TReR[Ti][spot-1];
    }
}
if(Simple == 1){
    add_seq_tree(Tj);
    add_seq_tree(Tk);
    del_seq_tree(Ti);
}
if( TNi[Tj] == 0 ) ID.push_back( Tj );
if( TNi[Tj] == 0){
    if(ReType == 0) PossRecom( Tj );
    if(RM == 1) PossMutRM( Tj );
}
if( TNi[Tk] == 0 ) ID.push_back( Tk );
if( TNi[Tk] == 0){
    if(ReType == 0) PossRecom( Tk );
    if(RM == 1) PossMutRM( Tk );
}
int n_j0 = TNi[Tj]; int n_k0 = TNi[Tk];
TNi[Ti] -= 1;
TNi[Tj] += 1;
TNi[Tk] += 1;
n_i = TNi[Ti];
n_j = TNi[Tj];
n_k = TNi[Tk];
N += 1;
VecSI CheckMuj , CheckMuk ;
if(RM == 0){
    for(itmp = 0 ; itmp <= L-1; ++itmp){
        if( THap[Ti][itmp] != 9) VMutation[itmp] -= THap[Ti][itmp] ;
    }
}
// Modifications of a and b ;
a -= Ta[Ti] ; b -= Tb[Ti] ;
a += Ta[Tj] ; b += Tb[Tj] ;
a += Ta[Tk] ; b += Tb[Tk] ;
if(ReType == 1){
    if(n_j == 1) AddNprime(Tj);
    if(n_k == 1) AddNprime(Tk);
}
if(RM == 0){
    for(itmp = 0 ; itmp <= L-1; ++itmp){
        if( THap[Tj][itmp] != 9){
            if( VMutation[itmp] == 0 && THap[Tj][itmp] == 1)
                CheckMuj.push_back( itmp );
            VMutation[itmp] += THap[Tj][itmp] ;
        }
        if( THap[Tk][itmp] != 9){
            if( VMutation[itmp] == 0 && THap[Tk][itmp] == 1)
                CheckMuk.push_back( itmp );
            VMutation[itmp] += THap[Tk][itmp] ;
        }
    }
}

```

```

    }
  }
}

// Events ;
if(n_i == 0) DeleteSandE(Ti) ; // Delete sequence i and associated events ;
if(n_i == 1) DeleteCoaId(Ti) ; // Delete CoaId event of sequence j ;
if(n_j == 1){
  for(sjk=ID.begin() ; sjk !=ID.end() ; ++sjk){
    if(*sjk == Tj || *sjk == Tk ) continue;
    PossCoaDi(*sjk , Tj);
  }
  if(CheckMuj.size() > 0) PossMut(CheckMuj); // Generate a new mutation event
;

}
if((n_j == 2) && (n_j0 == 1)) CheckRemoveMu( Tj ) ;
if((n_j == 2) && (Ti != Tj)) PossCoaId( Tj ) ;
if(n_k == 1){
  for(sjk=ID.begin() ; sjk !=ID.end() ; ++sjk){
    if(*sjk == Tk) continue ;
    PossCoaDi(*sjk , Tk);
  }
  if(CheckMuk.size() > 0) PossMut(CheckMuk); // Generate a new mutation event
;

}
if((n_k == 2) && (n_k0 == 1)) CheckRemoveMu( Tk ) ;
if((n_k == 2) && (Ti != Tk)) PossCoaId( Tk ) ;
}
}

// +-----+
// | Adjust n_j' and n_k' |
// +-----+
void Adjustnjpnkp(){
  Vec Poss1 , Poss2 ;
  int LastIs = -1 , Is , spot ;
  for(unsigned int Ev=0 ; Ev != PEvent.size() ; ++ Ev){
    if( PEvent[Ev][0] != 4) continue;
    Is = int(PEvent[Ev][1]) ;
    if(LastIs != Is){
      Poss1 = possible_ancestor_r( Is ) ;
      Poss2 = possible_ancestor_l( Is ) ;
      LastIs = Is ;
    }
    spot = PEvent[Ev][2] ;
    PEvent[Ev][3] = Poss1[spot-1] - TNi[Is] ;
    PEvent[Ev][4] = Poss2[L - spot - 1] - TNi[Is] ;
  }
}

//+-----+;
//| Getnjnk() |;
```

```

//+-----+;
void Getnjnk(){
    short int spot = PEvent[ChoosenEvent][2];
    int np = int(VNprime[spot-1]);
    // Choice of a sequence to recombine ;
    long double Alea = 1 ;
    if(np > 1) Alea = mtrand2.randInt(np-1) + 1;
    int ISeq = 1 ; Ti = -1; sjk = ID.begin();
    while(Ti == -1){
        if( ( spot >= TAnc[*sjk][0] ) && ( spot < TAnc[*sjk][1] ) ){
            if(ISeq == Alea) Ti = *sjk ;
            ++ISeq ;
        }
        ++sjk ;
    }
    // Creation of sequences j and k ;
    if(Simple == 0){
        Tj = TReL[Ti][spot-1];
        Tk = TReR[Ti][spot-1];
    }
    if(Simple == 1){
        Tj = SeqPosL(Ti);
        Tk = SeqPosR(Ti);
        if(Tj == -1) Tj = TReL[Ti][spot-1];
        if(Tk == -1) Tk = TReR[Ti][spot-1];
    }
    if(debug > 50) cout << "[Ti " << Ti << "]" [Tj " << Tj << " Tk = " << Tk << "]" << endl;
    n_j = TNi[Tj] ; n_k = TNi[Tk];
}

//+-----+;
//|      InsertGraph()      |;
//+-----+;
void InsertGraph(long double rt){
    int insert_pos = -1 ;
    int count = 0 ;
    while(insert_pos == -1){
        if(LikP > LikMax[count]) insert_pos = count ;
        count ++ ;
    }
    LikMax.insert( LikMax.begin() + insert_pos , LikP );
    LikStep.insert( LikStep.begin() + insert_pos , StepP );
    LikSeed.insert( LikSeed.begin() + insert_pos , GraphSeed );
    VRt.insert( VRt.begin() + insert_pos , rt );
    LikMax.erase( LikMax.begin() + NbGraphToKeep + 1 );
    LikStep.erase( LikStep.begin() + NbGraphToKeep + 1 );
    LikSeed.erase( LikSeed.begin() + NbGraphToKeep + 1 );
    VRt.erase( VRt.begin() + NbGraphToKeep + 1 );
    MinimumLik = LikMax[NbGraphToKeep-1];
}

```

```

//+-----+;
//|   CheckData() |;
//+-----+;
ListOfList CheckData(ListOfList Y){
    short int IS1 = -1 , IS2 = 1 ;
    for(i1=Y.begin() ; i1!=Y.end(); ++i1){
        IS1 += 1;
        IS2 = IS1 + 1;
        for(i2=i1+1 ; i2!=Y.end(); ++i2){
            if( (equal( (*i1).begin()+1, (*i1).end(), (*i2).begin()+1) == 1)
                && (Trait[IS1] == Trait[IS2])){
                ((*i1).begin()) += ((*i2).begin());
                ((*i2).begin()) = 0 ;
            }
            IS2 += 1;
        }
    }
    List SeqToRemove; IS1 = 0;
    for(i1=Y.begin() ; i1!=Y.end(); ++i1){
        if( ((*i1).begin()) == 0 ) SeqToRemove.push_back( IS1 );
        ++IS1 ;
    }
    IS1 = 0;
    for(j=SeqToRemove.begin() ; j !=SeqToRemove.end() ; ++j){
        Y.erase(Y.begin()+*j - IS1);
        Trait.erase(Trait.begin()+*j - IS1);
        IS1 ++ ;
    }
    return(Y);
}

//+-----+;
//|   TNTN(): TextNumber To Number |;
//+-----+;
int TNTN(string TNumber){
    int Number = 1 ;
    if(TNumber[TNumber.length()-1] == 'M') Number = 1000000;
    if(TNumber[TNumber.length()-1] == 'm') Number = 1000;
    if(TNumber[TNumber.length()-1] == 't') Number = 1000;
    if(TNumber[TNumber.length()-1] == 'm' || TNumber[TNumber.length()-1] ==
    'M'
        || TNumber[TNumber.length()-1] == 't') TNumber.resize(TNumber.size()-1);
    double FinalNumber = atof((ConvertSC2(TNumber))) ;
    FinalNumber *= Number ;
    int FNumber2 = arondi_di(FinalNumber);
    return(FNumber2);
}

//+-----+;
//|   Main() |;
//+-----+;
int main(int argc, char *argv[]){

```

```
/* interpretation of the command line */
InfoRestore = 0;
program_name = argv[0];
if(argc == 1){
    cerr <<" You should give at least a parameter file."<<endl;
    usage();
}
while((argc > 1) && (argv[1][0] == '-')){
    switch (argv[1][1]){
        case 'd':
            ++ argv;
            debug = atoi(&argv[1][0]);
            -- argc;
            break;
        case 'o':
            ++ argv;
            res_file_i = &argv[1][0];
            -- argc;
            break;
        case 'p':
            ++ argv;
            par_file = &argv[1][0];
            -- argc;
            break;
        case 'y':
            ++ argv;
            dat_file = &argv[1][0];
            -- argc;
            break;
        case 'r':
            ++ argv;
            RandomSeed = atoi(&argv[1][0]);
            -- argc;
            break;
        case 'R':
            ++ argv;
            InfoRestore = 1 ;
            -- argc;
            break;
        case 'N':
            ++ argv;
            NResamp_i = atoi(&argv[1][0]);
            -- argc;
            break;
        case 'G':
            ++ argv;
            NbGraphToKeep_i = TNTN(&argv[1][0]);
            -- argc;
            break;
        case 'e':
            ++ argv;
```

```

        proj_nbsim = TNTN(&argv[1][0]);
        -- argc;
        break;
    case 'g':
        ++ argv;
        plot = atoi(&argv[1][0]);
        -- argc;
        break;
    case 'm':
        ++ argv;
        muinit = atof(&argv[1][0]);
        -- argc;
        break;
    case 'P':
        ++ argv;
        Phase0 = atoi(&argv[1][0]);
        -- argc;
        break;
    case 'K':
        ++ argv;
        K = TNTN(&argv[1][0]);
        -- argc;
        break;
    case 'Z':
        ++ argv;
        rnd_file_i = &argv[1][0];
        -- argc;
        break;
    default:
        cerr << "Option " << argv[1] << " not recognized.\n";
        usage();
    }
    ++ argv;
    -- argc;
}

// Time and Random number generator ----- ;
Time StartTime ;
clock_t ticks = clock();
MTRand mtrand0; // automatically seed ;
if(RandomSeed == 0) RandomSeed = mtrand0.randInt(); // Initial Seed ;
mtrand0.seed( RandomSeed ) ;

// Generator of probabilities ;
MTRand::uint32 seed[ MTRand::N ];
for( int s = 0; s < MTRand::N; ++s )
    seed[s] = mtrand0.randInt() ; // fill with anything but all zeroes
mtrand1.seed( seed );
if(Phase == -1) mtrand2.seed( mtrand1.randInt() );

// Parameters ----- ;
LireParm(); // Read Parameter file ;

```

```

if(res_file_i) res_file = res_file_i;
if(rnd_file_i) rnd_file = rnd_file_i;
// String FileR: nom du chichier resultats sans l'extension ;
FileR = (string)(res_file);
unsigned int LFileR = FileR.length();
if(LFileR >= 4){
    if(FileR[LFileR-1] == 't' && FileR[LFileR-2] == 'x' && FileR[LFileR-3] ==
't')
        FileR.resize( LFileR -4);
}
//if(rnd_file_i) {} else rnd_file = FileR.c_str();
if( ! rnd_file_i) rnd_file = ConvertSC2(FileR);
if(NbGraphToKeep_i != -1) NbGraphToKeep = NbGraphToKeep_i ;

if(Phase0 > -1) Phase = Phase0 ;
if(K>0) NbSim = K;
if(NbSim < 10) PrintNbSim = 1;
if(PrintNbSim == -1) PrintNbSim = NbSim / 10 ;

// Data ----- ;
ListOfList Y0 = LireData(); // Read Data ;
Y0 = CheckData(Y0); // Check if some haplotypes are repeated ;
Y.assign(Y0.begin(), Y0.end());
NT = Y0.size(); // Nb of differents types (d) ;
if(kappa == 0) VPS = 0 ; // Variable Population Size ;
else if(kappa > 0) VPS = 1 ;

// Parameters consistency ----- ;
if(Phase != -1 && NResamp == -1){
    NResamp = arondi_di(N0 / 10 ) ;
    if(NResamp == 0) NResamp = 2;
}
if(NResamp_i != -1) NResamp = NResamp_i ;
if(Phase == 0 && NbGraphToKeep == -1){
    if(NbSim > 1000000) NbGraphToKeep = int(NbSim / 1000 ) ;
    else NbGraphToKeep = int(NbSim / 100 ) ;
    if(NbGraphToKeep == 0) NbGraphToKeep = 1;
}

// Output of information in debug mode ----- ;
#ifdef DEBUG
if(debug > 1) cout << "+ Parameters\n";
if(debug > 1) cout << " [Ne "<<Ne<<"] [ K="<< NbSim<<"] [ NbCan="<<NbCan<<"]
[Simple "<<Simple<<"]\n";
if(debug > 1) PrintPL(Mu0, "Mu0");
if(debug > 1) PrintPL(Dist0, "Dist0");
if(debug > 1) cout << " Data File: "<< dat_file << " / Paramater File: " << par_file
<< " \n";
if(debug > 1) cout << " Debug Level set to " << debug << "\n";
if(debug > 1) cout << " VariablePopulationSize: "<<VPS<<" /
kappa="<<kappa<<endl;
if(debug > 1) cout << " d = " << NT << " / " << "L = " << NM << "\n";

```

```

if(debug > 1) PrintGL(Y0, "Haplotypes");
if(debug > 1) PrintPL(Trait, "Trait");
if(debug > 1) cout << "Sample Size: " << N0 << "\n";
if(debug > 0){
    if(Phase == 0){
        cout << "[Phase: Search] [NbGraphToKeep: "<<NbGraphToKeep<<"]\n";
    }
    if(Phase == 1){
        cout << "[Phase: Resampling] [Replicates: "<<R<<"]\n";
    }
}
#endif

// Driving Values ----- ;
VecD EmptyDriving;
VecD Coordinate;
Coordinate.push_back(0);
double GCoor = 0;
for(int itmp=1 ; itmp!= NM; ++itmp){
    GCoor += Dist0[itmp-1];
    Coordinate.push_back(GCoor*100);
}
for(int itmp=1 ; itmp!= NM; ++itmp){
    Driving.push_back(-1);
}
if(DrivingInitial.size() > 0){
    for(unsigned int itmp=0 ; itmp!= DrivingInitial.size()/2+1; itmp += 2){
        double vartmp = DrivingInitial[itmp+1] ; /*- Coordinate[DrivingInitial[itmp]-1];
        if(DrivingInitial[itmp+1] < Coordinate[int(DrivingInitial[itmp]-1)] ||
           DrivingInitial[itmp+1] > Coordinate[int(DrivingInitial[itmp])]) {
            cerr << "Problem with Driving Values: outside admissible range."<<endl;
            exit(8);
        }
        Driving[int(DrivingInitial[itmp]-1)] = vartmp ;
    }
}

// Distances of recombinations... ----- ;
if ( int(Dist0.size()) != NM-1){
    cerr << "ERROR 1: Number of distances doesn't correspond "
         << "to number of markers minus one.\n";
    exit(2);
}
r = 0.0;
ListD : : iterator k;
for(k=Dist0.begin(); k != Dist0.end(); ++k) {
    r = r + *k;
}

// Calculus of the number of candidates for local IS ----- ;
if(VNbCan.size() == 0) { // Nb of candidates per interval not known
    VNbCan.clear();
}

```



```

    for(k=Dist0.begin(); k!= Dist0.end(); ++k){
        VNbCan.push_back( arondi_di( (double)( *k / r * NbCan) ) );
    }
}
NbCan = 0;
for(sj=VNbCan.begin(); sj!= VNbCan.end(); ++sj){
    if(*sj != 0 ) {
        if( *sj % 2 == 0)
            *sj = *sj + 1;
    }
    NbCan += *sj;
}

// Output of information in debug mode ----- ;
#ifdef DEBUG
if(debug > 1) cout << "r = " << r << ". ";
if((MarkerT == 1) && (debug > 1) ) cout << "Marker Type: SNP\n";
if((MarkerT == 0) && (debug > 1)) cout << "Marker Type: Micro Sat.\n";
if(debug > 2) cout<< "NbCan=" << NbCan <<"\n";
if(debug > 2) PrintV(VNbCan,"VNbCan");
if(debug > 2){
    cout <<"[ TraceRt ="<<TraceRt<<" / FreqTraceRt = "<< FreqTraceRt<<"]"<<endl;
if(debug > 2) cout << "[Simple "<<Simple+1-1<<" | In "<<In<<]"<<endl;
}
#endif

// Initialisation of some vectors ----- ;
for(int index = 1; index <= NbCan; ++index){
    LikE.push_back(0);
    LikM.push_back(0);
}

for(unsigned int index=1 ; index != Dist0.size()+1 ; ++ index)
Abandon.push_back(0);
unsigned short int interval = 0;
unsigned int LastCan = 0; // Index debut dans ProbF ;
unsigned int FirstCan = 0;
unsigned int RealCan;

VecE ProdFE ;
unsigned int NbCanL ;

// Creation of tables of information to use later ----- ;
CreateTable(NM);
#ifdef DEBUG
if(debug > 199){
    PrintVV(THap,"THap");
    PrintV(Ta,"Ta");
    PrintVV(TMu,"TMu");
    PrintVV(TAnc,"TAnc");
    PrintVV(TReR,"TReR");
    PrintVV(TReL,"TReL");
}

```

```

    PrintVV(TCoa,"TCoa");
}
#endif
for(k=Dist0.begin(); k!= Dist0.end(); ++k){
    SumWi.push_back(0);
    SumWi2.push_back(0);
    ESS.push_back(0);
    VSNbR.push_back(0);
    VS2NbR.push_back(0);
    VSNbM.push_back(0);
    VS2NbM.push_back(0);
    VSNbC.push_back(0);
    VS2NbC.push_back(0);
    VSNbE.push_back(0);
    VS2NbE.push_back(0);
    VNbGraphDone.push_back(0);
}

// Empty the contents of RndFile;
if(Phase == 0){
    string FileRND = (string)(rnd_file) + ".rnd";
    const char* rnd_file = FileRND.c_str() ;
    ofstream outR;
    outR.open(rnd_file, ios::trunc);
    outR.close();
}

// Restore if crash;
if(InfoRestore == 1 ) Restore();

// +-----+
// |   For each interval in the sequence   |
// +-----+
for(k=Dist0.begin(); k!= Dist0.end(); ++k){
    interval ++;

    // Create a vector of candidates coordonate ----- ;
    VCR.clear();
    long double rglobal = 0 ;
    for(unsigned int itemp3 = 1 ; itemp3 != Dist0.size() + 1 ; ++ itemp3){
        double runit = Dist0[itemp3-1] / (VNbCan[itemp3-1] + 1);
        for(int itmp2 =1; itmp2 != VNbCan[itemp3-1] +1; ++ itmp2){
            rglobal += runit ;
            VCR.push_back(100*rglobal);
        }
        rglobal += runit ;
    }

#ifdef DEBUG
    if(debug>2) PrintV(VCR,"VCR");
    if(debug>2) PrintV(DrivingInitial,"DrivingInitial");
    if(debug>2) PrintV(Driving,"Driving");

```

```

#endif

// Phase: search best graphs, or resmaple ----- ;
if(Phase == 0){
  NbAncestors = NResamp ;
  MinimumLik = 0 ;
  if(NbSim < NbGraphToKeep) NbGraphToKeep = NbSim;
    LikMax.clear();
    LikStep.clear();
    VRt.clear();
    LikSeed.clear();
    MinLik = 0 ;
    for(int itemp=0 ; itemp != NbGraphToKeep ; ++ itemp){
      LikMax.push_back(0);
      LikStep.push_back(0);
      VRt.push_back(0);
      LikSeed.push_back(0);
    }
  }
  //int total = 0;
  Vec RNbSim ;
  if(Phase == 1 && VNbCan[interval-1] > 0){
    LireRnd(interval) ;
    EXTNUM SommeS = 0;
    for(sje=LikVal.begin() ; sje !=LikVal.end() ; ++sje) SommeS += *sje ;
    for(sje=LikVal.begin() ; sje !=LikVal.end() ; ++sje) *sje /= SommeS ;
    for(sje=LikVal.begin() ; sje !=LikVal.end() ; ++sje)
      RNbSim.push_back( arondi_di( (double)(*sje * NbSim)) ) ;
    NbSim = 0;
    for(sj=RNbSim.begin() ; sj !=RNbSim.end() ; ++sj)
      NbSim += *sj ;
    VNbSim.push_back(NbSim);
  }
  if(Phase == 1 && VNbCan[interval-1] == 0) VNbSim.push_back(0);

  // Initialisations... ----- ;
  NbCanL = VNbCan[interval-1] ; // Nb can local ;
  if( NbCanL == 0 ) continue ; // Drop to the next interval ;
  FirstCan = LastCan + 1; // First candidate of the current interval ;
  LastCan = NbCanL + FirstCan - 1; // Last candidate of the current interval ;
  RealCan = (LastCan - FirstCan) / 2 + FirstCan;
  VTraceRt.clear(); VTraceRtS.clear(); VTraceRtL.clear();

  VecE TmpE;
  if(Phase == 1){
    // Likelihood ;
    for(int itmp = FirstCan-1 ; itmp != LastCan ; ++ itmp){
      TmpE.push_back(LikM[itmp]);
    }
  }
  double driving_value = -1 , driving_initial = Driving[interval-1];

```

```

int driving_number = -1 ;
if(driving_initial > -1){
    for(unsigned int itmp=FirstCan; itmp != LastCan ; ++itmp){
        if(driving_initial > VCR[itmp-1] && driving_initial < VCR[itmp]){
            if((driving_initial-VCR[itmp-1]) < (VCR[itmp]-driving_initial)){
                driving_value = VCR[itmp-1];
                driving_number = itmp-1 ;
            }
            else{
                driving_value = VCR[itmp];
                driving_number = itmp ;
            }
        }
    }
    RealCan = driving_number ;
}

if(driving_value == -1){
    driving_value = VCR[ RealCan-1 ] ;
    driving_number = RealCan - FirstCan + 1 ;
}
else driving_number = RealCan - FirstCan + 2 ;
if(debug > 100) cout <<"Driving value="<<driving_value<<" (no
"<<driving_number<<)"<<endl;
long double rl0 = ( driving_value - Coordinate[interval-1] ) / 100;
long double rr0 = ( Coordinate[interval] - driving_value ) / 100 ;
VecD LeftDist , RightDist;
for(unsigned int itmp=FirstCan-1; itmp !=LastCan ; ++itmp){
    LeftDist.push_back((VCR[itmp]-Coordinate[interval-1])/100);
    RightDist.push_back((Coordinate[interval] - VCR[itmp])/100);
}
Driving[interval-1] = driving_value ;
#ifdef DEBUG
if(debug>2) PrintV(Driving,"Driving");
if(debug > 100) cout <<"Driving Value: "<< driving_value <<" RealCan=" <<
RealCan
    << "DrivingNumber=" << driving_number << endl;
if(debug > 100) PrintV(LeftDist,"LeftDist");
if(debug > 100) PrintV(RightDist,"RightDist");
#endif

// Restore from crash... ----- ;
if(InfoRestore == 1 && interval < intervalRestore){
    cout << "Interval "<<interval<<" "<<"["<<NbSim<<"] Restored."<<endl;
    continue;
}

// Transformation of data and distances ----- ;
Y1.clear();
Y1.assign(Y0.begin(), Y0.end());
Dist.clear();

```

```

Dist.assign(Dist0.begin(), Dist0.end());
ProdFE.push_back(0); // Assure que la liste ne soit pas vide ;

short int index = -1;
long double mu = -1 ;
if(Mu0.size() > 0) mu = *(Mu0.begin() + 0);
else{
    cerr <<" --> mu is not defined. I assume mu = rho"<<endl;
    mu = 4*Ne*r;
}
if(muinit > 0 ) mu = muinit ; //Muinit in the command line ;

rho = 4*Ne*r;
theta = 4*Ne*mu;
for(i=Y1.begin(); i!= Y1.end(); ++i){
    index ++;
    if(Trait[index] == 0)
        (*i).insert((*i).begin() + interval + 1, 1, 0);
    if(Trait[index] == 1)
        (*i).insert((*i).begin() + interval + 1, 1, 1);
    if(Trait[index] == 9)
        (*i).insert((*i).begin() + interval + 1, 1, 9);
}
Dist.insert(Dist.begin() + interval, 1, Dist0[interval-1]/2);
Dist[interval-1] = rl0 ;
Dist[interval] = rr0 ;

CreateTableTB();

// Output of information in debug mode ----- ;
#ifdef DEBUG
if(debug > 10) cout << "\n +======"
    <<"====="
    << "====="
<< "\n";
if(debug > 10) cout << " | Interval " << interval << "\n";
if(debug > 10) cout << "\n +======"
    <<"====="
    << "====="
<< "\n\n";
if(debug > 20) cout <<"FirstCan: " <<FirstCan<<" LastCan: " <<LastCan<<"
RealCan: "
    <<RealCan<<" \n";
if(debug > 20) cout <<"rl0:" <<rl0<<" rr0:" <<rr0<<"\n";
if(debug > 20) PrintPL(Dist, "Dist modifiee");
#endif

// Identification of sequences from THap ;
ID.clear();
// Creation of TNi -----;
TNi.clear();
for(si=THap.begin() ; si!=THap.end(); ++si) TNi.push_back(0);

```

```

Vec Seq;
for(i=Y1.begin() ; i != Y1.end() ; ++i){
    Seq.clear();
    int ni = *((*i).begin());
    Seq.assign((*i).begin()+1 , (*i).end());
    int Ti = NumSeq(Seq);
    ID.push_back( NumSeq(Seq) );
    TNi[Ti] += ni;
}

L = NM + 1 ; // Nb de loci;
if(debug > 50) PrintData();

// Find frequencies of each allele ;
Vec Totaux ;
FreqAA.clear();
for(short int IM=0 ; IM <= L-1 ; ++IM){
    FreqAA.push_back(0);
    Totaux.push_back(0);
}
for(sjk=ID.begin() ; sjk !=ID.end() ; ++sjk){
    for(short int IM=0 ; IM <= L-1 ; ++IM){
        if(THap[*sjk][IM] == 0) FreqAA[IM] += TNi[*sjk];
        if(THap[*sjk][IM] != 9) Totaux[IM] += TNi[*sjk];
    }
}
for(short int IM=0 ; IM <= L-1 ; ++IM){
    FreqAA[IM] /= Totaux[IM] ;
}
if(debug > 2) PrintV(FreqAA,"FreqAA");

// Construt the tree ;
if(Simple == 1){
    FirstNode.zeroTree = NULL ;
    FirstNode.oneTree = NULL ;
    FirstNode.nineTree = NULL ;
    FirstNode.n_b = 0 ;
    FirstNode.n_e = 0 ;
    // Empty the tree;
    delete_tree();
    // Initilize the tree ;
    add_one = 0 ;
    for(sjk=ID.begin() ; sjk !=ID.end() ; ++sjk){
        add_seq_tree(*sjk);
    }
    add_one = 1 ;
}

// Checks off all possibles events;
PEvent.clear(); Prob.clear();

```

```

// Coalescence of identical types sequences;
for(sjk=ID.begin() ; sjk !=ID.end() ; ++sjk){
    PossCooId(*sjk);
}

// Coalescence of differents types sequences;
for(sjk1=ID.begin() ; sjk1 !=ID.end() ; ++sjk1){
    for(sjk2=sjk1+1 ; sjk2 !=ID.end() ; ++sjk2){
        PossCooDi(*sjk1 , *sjk2);
    }
}

// Singleton Mutations ;
if(RM == 0){
    VMutation.clear();
    VecSI MuTmp;
    for(short int IM=0 ; IM <= L-1 ; ++IM){
        int Somme = 0;
        for(sjk=ID.begin() ; sjk != ID.end() ; ++sjk){
            if(THap[*sjk][IM] != 9 )
                Somme += TNi[*sjk] * THap[*sjk][IM] ;
        }
        VMutation.push_back(Somme);
        if(Somme == 1) MuTmp.push_back(IM);
    }
    if(MuTmp.size() > 0) PossMut(MuTmp);
}
else{
    for(sjk=ID.begin() ; sjk!=ID.end() ; ++sjk)
        PossMutRM(*sjk);
}

// Recombinations ;
if(ReType == 1){
    VNprime.clear();
    for(int Spot=1 ; Spot != L; ++Spot){
        int np = 0 ;
        for(sjk=ID.begin() ; sjk!=ID.end() ; ++sjk){
            if( (Spot >= TAnc[*sjk][0]) && (Spot < TAnc[*sjk][1]) ) np += 1;
        }
        VNprime.push_back(np);
        VecSI TEvent;
        TEvent.push_back(4);
        TEvent.push_back(0);
        TEvent.push_back(Spot);
        PEvent.push_back(TEvent);
        Prob.push_back(0);
    }
}
if(ReType == 0){
    for(sjk=ID.begin() ; sjk!=ID.end() ; ++sjk)
        PossRecom(*sjk);
}

```

```

}

// Calculus of a and b ;
a = 0; b = 0;
for(sjk=ID.begin() ; sjk != ID.end(); ++sjk){
    a += TNi[*sjk] * Ta[*sjk];
    b += TNi[*sjk] * Tb[*sjk];
}

PEvent0.assign(PEvent.begin(),PEvent.end());
Prob0.assign(Prob.begin(),Prob.end());
VMutation0.assign(VMutation.begin(),VMutation.end());
VNprime0.assign(VNprime.begin(),VNprime.end());
ID0.assign(ID.begin(),ID.end());
TNi0.assign(TNi.begin(),TNi.end());
a0 = a ;
b0 = b ;

int g = 1 ;

if(Phase == 1){
    R = RNbSim[g-1] ;
    replicate = 0;
}
int iter;
NbGraphDone = 0;

// +-----+
// | For a given interval, do NbSim graphs |
// +-----+
for(int Sim=1; Sim != NbSim +1 ; Sim++){
    N = N0 ;

    if(Phase == 1){
        replicate += 1 ;
        if(replicate == R+1 ){
            replicate = 1 ;
            g += 1;
        }
    }
}

ProdFE.clear();

char *prefix = "";
for(unsigned int index=1 ; index <= NbCanL ; ++ index)
    #ifdef Extnum
    ProdFE.push_back( 1 );
    #endif
    #ifndef Extnum
    ProdFE.push_back( lambda );
    #endif
iter = 0;

```



```

if(Sim == 1) {
    float K2 = NbSim;
    if(K2>=1000000){
        K2 /= 1000000;
        prefix = "M";
    }
    else if(K2>=1000){
        K2 /= 1000;
        prefix = "m";
    }
    cerr <<"Interval " <<interval<<" [" <<K2<<prefix<<"] Sim ";
}
if( Sim%PrintNbSim == 0) {
    float K2 = Sim;
    if(K2>=1000000){
        K2 /= 1000000;
        prefix = "M";
    }
    else if(K2>=1000){
        K2 /= 1000;
        prefix = "m";
    }
    cerr<<K2<<prefix<<" ";
    BackUp(Sim-1 , interval );
}
if(Sim==NbSim) cerr <<"\n";
if(InfoRestore == 1 && interval == intervalRestore && Sim<Sim.Restore+1)
continue;
ID.assign(ID0.begin(),ID0.end());
TNi.clear();
TNi.assign(TNi0.begin(),TNi0.end());
VNprime.assign(VNprime0.begin(),VNprime0.end());
PEvent.assign(PEvent0.begin(),PEvent0.end());
Prob.assign(Prob0.begin(),Prob0.end());
VMutation.assign(VMutation0.begin(),VMutation0.end());
a = a0 ;
b = b0 ;

c_time = 0 ;

if(Phase == 0){
    GraphSeed = mtrand1.randInt();
    mtrand2.seed(GraphSeed);
}
if(Phase == 1 && replicate == 1){
    mtrand2.seed(LikSeed[g-1]);
    PrintNbSim = NbSim / 10;
    if(PrintNbSim < 1) PrintNbSim = 1;
}
if(Phase == 1 && replicate > 1){
    ID.assign(IDbackup.begin(),IDbackup.end());
    TNi.assign(TNibackup.begin(),TNibackup.end());
}

```

```

VNprime.assign(VNprimebackup.begin(), VNprimebackup.end());
PEvent.assign(PEventbackup.begin(), PEventbackup.end());
Prob.assign(Probackup.begin(), Probackup.end());
VMutation.assign(VMutationbackup.begin(), VMutationbackup.end());
a = abackup ;
b = bbackup ;
ProdFE.assign(ProdFEbackup.begin(), ProdFEbackup.end());
N = Nbackup;
iter = iterbackup ;
mtrand2.seed(mtrand3.randInt());
}

if(Simple == 1){
    // Empty the tree;
    delete_tree();
    // Initilize the tree ;
    add_one = 0 ;
    for(sjk=ID.begin() ; sjk !=ID.end() ; ++sjk){
        add_seq_tree(*sjk);
    }
    add_one = 1 ;
}
NbR = 0 ; NbC = 0 ; NbE = 0; NbM = 0;
Vec InfoN;
for(int index=1 ; index <= N-1 ; ++index) InfoN.push_back(0);
converge = 1;
NotTheMRCA = 0;
if(debug>140) cout<<"[SIM "<<Sim<<" ";
if(debug>140) PrintV(Like, "Like");

// +-----+
// |      Construction of a graph      |
// +-----+
while( N > NbAncestors ) {

    ++ iter ;

#ifdef DEBUG
    if(debug == -1 || debug > 499){
        VecOfVecD Listtmp;
    }
#endif

#ifdef DEBUG
    if(debug>52) cout<<"\n[Sim "<<Sim<<" [Int "<<interval<<" [Iter "<<iter
        <<" [N="<<N<<"]\n";
    if(debug>52) cout << "-----\n";
#endif
    beta = b / (N * r);
    alpha = a / ((long double)(N * L) );

    if(VPS == 1) {

```

```

    c_time = TimeNextEvent2();
    lambda_t = exp( kappa * c_time );
}

if( iter >= MaxIter || converge == 0 ){
    Abandon[interval-1] += 1 ;
    iter = MaxIter + 1 ; // to no count the graph ;
    sje = ProdFE.begin();
    for(unsigned int ican=1 ; ican <= NbCanL ; ++ ican){
        *sje = 0 ;
        ++sje;
    }
    break ;
}

#ifdef DEBUG
if(debug > 50) PrintData();
if((debug > 50) && (RM == 0)) PrintV(VMutation,"VMutation");
if(debug > 50 && ReType == 1) PrintV(VNprime,"VNprime");
#endif

// Calculus of probabilities for each event ;
if(Simple == 1) Adjustnjpnpk();

CalculProb();

// Choose an event ;
ChooseEvent();

if( TypeEv == 1 || TypeEv == 2) ++NbC ;
if( TypeEv == 4 ) ++NbR ;
if( TypeEv == 3 ) ++NbM ;

#ifdef DEBUG
if(debug > 50) PrintPEvent();
if(debug > 35) cout << "[a=" << a << "]" [alpha=" << alpha << "]" [b=" << b << "]"
[beta="
                << beta << "]" \n";
if(debug > 32){
    VecSI Even = PEvent[ChoosenEvent];
    if( TypeEv == 1 ){
        cout << "--> Coalescence of 2 seq. of types "
            << Even[1] << " (Ev no " << ChoosenEvent << ") [p="
            << Prob[ChoosenEvent]/SommeP << "]" ;
    }
    if( TypeEv == 2 ){
        cout << "--> Coalescence of seq. "
            << Even[1] << " and " << Even[2] << " into seq. "
            << Even[3] << " (Ev no " << ChoosenEvent << ") [p="
            << Prob[ChoosenEvent]/SommeP << "]" ;
    }
    if( TypeEv == 3 ){
        cout << "--> Mutation of seq. "

```

```

        << Even[2] << ", marker " << Even[1] << " into seq "
        << Even[3] << " (Ev no " << ChoosenEvent <<") [p="
        <<Prob[ChoosenEvent]/SommeP<<" ] ";
    }
    if( TypeEv == 4 ){
        if( ReType == 1 ){
            cout << "--> Re in interval "
                << Even[2] << " (Ev no " << ChoosenEvent <<") [p="
                <<Prob[ChoosenEvent]/SommeP<<" ] ";
        }
        if( ReType == 0 ){
            cout << "--> Re of Seq "<<Even[1]<<" in interval "
                << Even[2] << " (Ev no " << ChoosenEvent <<") [p="
                <<Prob[ChoosenEvent]/SommeP<<" ] ";
        }
    }
}
#endif

if( VPS == 0 )
    D = N*(N-1 + alpha*theta + beta*rho);
if( VPS == 1 )
    D = N*((N-1)*lambda_t + alpha*theta + beta*rho);
if( TypeEv == 4 && ReType == 0 ){
    VecSI Even = PEvent[ChoosenEvent];
    F_XY = SommeP / ( D * ReWeight ) ;
}

if( TypeEv == 4 && ReType == 1 ){
    Getnjk();
    F_XY = ((n_j + 1)*(n_k + 1)*SommeP) / ( D * ReWeight ) ;
}

if( TypeEv != 4 ) F_XY = SommeP / D ;
if( TypeEv == 4 ){
    int spot = PEvent[ChoosenEvent][2] ;
    if( interval == spot ){
        for( unsigned int ican=0 ; ican < NbCanL ; ++ ican )
            ProdFE[ican] *= F_XY * ( LeftDist[ican] / rl0 ) ;
    }
    else if( interval + 1 == spot ){
        for( unsigned int ican=0 ; ican < NbCanL ; ++ ican )
            ProdFE[ican] *= F_XY * ( RightDist[ican] / rr0 ) ;
    }
    else{
        for( unsigned int ican=1 ; ican <= NbCanL ; ++ ican )
            ProdFE[ican-1] *= F_XY ;
    }
}
else{
    for( unsigned int ican=1 ; ican <= NbCanL ; ++ ican )
        ProdFE[ican-1] *= F_XY ;
}
}

```

```

#ifdef DEBUG
if( debug > 5) cout<<"S="<<SommeP<<" D="<<D<<" F="<<F_XY<<endl;
if( debug > 2) PrintV(ProdFE,"ProdF");
#endif

#ifdef Extnum
if(ProdFE[driving_number-1] < 1e-320) converge = 0 ;
#endif

UpdateY();

#ifdef DEBUG
if( VPS == 1 && debug > 5) cout << "[T="<<c_time
<<"] [lambda="<<lambda_t<<"] [N(t)="<<Ne / lambda_t<<"]\n";
#endif

if(Phase == 1 && replicate == 1){
  if(InfoN[N-1] == 0){
    InfoN[N-1] = 1;
    if( N == NResamp ){
      IDbackup.assign(ID.begin(), ID.end());
      TNibackup.assign(TNi.begin(), TNi.end());
      VNprimebackup.assign(VNprime.begin(), VNprime.end());
      PEventbackup.assign(PEvent.begin(), PEvent.end());
      Probbackup.assign(Prob.begin(), Prob.end());
      VMutationbackup.assign(VMutation.begin(), VMutation.end());
      abackup = a ;
      bbackup = b ;
      Nbackup = N;
      iterbackup = iter;
      ProdFE.assign( TmpE.begin(), TmpE.end());
      ProdFEbackup.assign(ProdFE.begin(), ProdFE.end());
      mtrand2.seed(mtrand3.randInt());
    }
  }
}

if(Phase == 0){
  if( N == NResamp && InfoN[N-1] == 0){
    InfoN[N-1] = 1;
  }
}

} // END[---while(N>1 || iter <= MaxIter)---] ----- ;
// End of construction of a graph ----- ;

if(iter < MaxIter && Phase != 0 && TNi[0] !=1 ) NotTheMRCA = 1;

if(Phase == 0){
  StepP = iter ;
  LikP = ProdFE[ driving_number - 1 ] ;
}

```

```

        if( LikP > MinimumLik ) {
            InsertGraph(0) ;
        }
    }
    int ican = 0 ;

    for(unsigned int index3 = FirstCan-1; index3 <= LastCan-1; ++index3){
        if(iter < MaxIter) LikE[index3] += ProdFE[ican] ;
        ++ican ;
    }

    EXTNUM Ptmp = ProdFE[driving_number-1]/lambda ;
    SumWi[interval-1] += Ptmp ;
    SumWi2[interval-1] += pow(Ptmp,2) ;
    if(iter < MaxIter ){
        NbE = iter ;
        VNbGraphDone[interval-1] += 1 ;
        VSNbR[interval-1] += NbR ;
        VS2NbR[interval-1] += NbR*NbR ;
        VSNbM[interval-1] += NbM ;
        VS2NbM[interval-1] += NbM*NbM ;
        VSNbC[interval-1] += NbC ;
        VS2NbC[interval-1] += NbC*NbC ;
        VSNbE[interval-1] += NbE ;
        VS2NbE[interval-1] += NbE*NbE ;
    }
    if(TraceRt == 1 && iter < MaxIter){
        if( Sim%FreqTraceRt == 0){
            f_MaxLik(Sim) ;
        }
    }
} // END[---for(int Sim=1; Sim != Nsim; ++Sim)---]

if(Phase == 0 ) SaveIntermediateResult(interval) ;

} // END[---for(k=Dist0.begin(); k!= Dist0.end(); ++k)---]

for(sje = LikE.begin() ; sje != LikE.end(); ++sje) *sje /= NbSim ;

// Phase 0: save partial likelihood ;
if(Phase == 0){
    string FileRND = (string)(rnd_file) + ".rnd";
    const char* rnd_file = FileRND.c_str() ;
    ofstream outR;
    outR.open(rnd_file, ios::app);
    outR <<"L ";
    for(sje=LikE.begin(); sje != LikE.end() ; ++sje){
        outR << *sje / lambda << " ";
    }
    outR << endl;
}
}

```

```

// Phase 1: Mean number of iterations per interval ;
int nbtmp =0; EXTNUM sommetmp =0;
for(sj = VNbSim.begin() ; sj != VNbSim.end() ; ++sj){
    sommetmp += *sj;
    if(*sj != 0) nbtmp += 1;
}
MeanNbSim = arondi(sommetmp / nbtmp) ;

// ESS ;
for(unsigned int index=0 ; index != ESS.size() ; ++index){
    if(SumWi2[index] != 0) ESS[index] = ( pow(SumWi[index], 2) ) /
SumWi2[index] ;
    else ESS[index] = 0;
}

for(unsigned int index=0 ; index != VSNbR.size() ; ++index){
    if(VNBGraphDone[index] != 0){
        VSNbR[index] /= VNBGraphDone[index] ;
        VSNbM[index] /= VNBGraphDone[index] ;
        VSNbE[index] /= VNBGraphDone[index] ;
        VSNbC[index] /= VNBGraphDone[index] ;
        VS2NbR[index] /= VNBGraphDone[index] ;
        VS2NbC[index] /= VNBGraphDone[index] ;
        VS2NbM[index] /= VNBGraphDone[index] ;
        VS2NbE[index] /= VNBGraphDone[index] ;
        VS2NbR[index] -= VSNbR[index]*VSNbR[index] ;
        VS2NbM[index] -= VSNbM[index]*VSNbM[index] ;
        VS2NbE[index] -= VSNbE[index]*VSNbE[index] ;
        VS2NbC[index] -= VSNbC[index]*VSNbC[index] ;
    }
}

// Research of the minimum and maximum of Lik[] ;
MinLik = *(LikE.begin()+0) ;
MaxLik = *(LikE.begin()+0) ;
int IndexMax = 0 , IndexMin = 0 , itmp = -1;
for(sje=LikE.begin()+1 ; sje != LikE.end() ; ++sje){
    ++ itmp ;
    if(*sje < MinLik){
        IndexMin = itmp + 1;
        MinLik = *sje ;
    }
    if(*sje > MaxLik){
        IndexMax = itmp +1;
        MaxLik = *sje ;
    }
}

LikR.assign(LikE.begin() , LikE.end());

for(sje=LikR.begin() ; sje != LikR.end() ; ++sje){
    *sje /= LikE[IndexMax] ;
}

```

```

}

// Processing Time ----- ;
Time EndTime;
string stmp1 = StartTime.ascii();
string stmp2 = EndTime.ascii();
stmp1.resize(stmp1.length()-1);
stmp2.resize(stmp2.length()-1);

long double elapsed_time = (long double)(clock() - ticks);
CpuTime = SecondsToTime2(long int)(elapsed_time / CLOCKS_PER_SEC) ;
RealTime = SecondsToTime2(long int)(EndTime.delta(&StartTime));
cout << "[CPU " << CpuTime << ". / ";
cout << " Real time: " << RealTime << ".]" << endl;
cout << "[ " << stmp1 << " / " << stmp2 << "]" << endl;

WriteResult(Y0 , VNbCan , Dist0 , elapsed_time , IndexMax , EndTime ,
StartTime );
if(plot == 1 ) WritePlot(VNbCan, Dist0 );
if(TeXOutput == 1 )
    OutputTeX( VNbCan , Dist0 , elapsed_time , IndexMax , EndTime , StartTime
);

// Estimate of time to do x simulations ----- ;
if(proj_nbsim > 0) {
    long double unit_time = (elapsed_time / CLOCKS_PER_SEC ) / (NbSim) ;
    int tot_time = int(unit_time* proj_nbsim) ;
    cout << " + Total estimated time for " ;
    double proj_nbsim2 ;
    string prefix = "";
    if(proj_nbsim>=1000000){
        proj_nbsim2 = (double)(proj_nbsim) / 1000000 ;
        prefix = "M" ;
    }
    else if(proj_nbsim>=1000){
        proj_nbsim2 = (double)proj_nbsim / 1000 ;
        prefix = "m" ;
    }
    if(proj_nbsim<1000) proj_nbsim2 = proj_nbsim ;
    cout << proj_nbsim2 << prefix << " simulations is " << SecondsToTime2(tot_time) <<
"." << endl;
}

return(0);
}

```