Université de Montréal

Étude algorithmique et combinatoire de la méthode de Kemeny-Young et du consensus de classements

par

Robin Milosz

Département d'informatique et de recherche opérationnelle
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 informatique

octobre 2018

RÉSUMÉ

Une permutation est une liste qui ordonne des objets ou des candidats en fonction d'une préférence ou d'un critère. Des exemples sont les résultats d'un moteur de recherche sur l'internet, des classements d'athlètes, des listes de gènes liés à une maladie données par des méthodes de prédiction ou simplement des préférences d'activités à faire pour la prochaine fin de semaine. On peut être intéressé à agréger plusieurs permutations pour en obtenir une permutation consensus. Ce problème est bien connu en science politique et plusieurs méthodes existent pour agréger des permutations, chacune ayant ses propriétés mathématiques. Parmi ces méthodes, la méthode de Kemeny-Young, aussi nommée la médiane de permutations, permet de trouver un consensus qui minimise la somme des distances entre ce consensus et l'ensemble de permutations. Cette méthode détient plusieurs propriétés désirables. Par contre, elle est difficile à calculer, ouvrant par ce fait, la voie à de nombreux travaux de recherche. Une généralisation de ce problème permet de considérer les classements qui contiennent des égalités entre les objets classés et qui peuvent être incomplets en ne considérant qu'un sous-ensemble d'objets. Dans cette thèse nous étudions la méthode de Kemeny-Young sous différents aspects :

— Premièrement, une réduction d'espace de recherche est proposée. Elle permet d'améliorer les temps de calcul d'approches exactes pour le problème.
— Deuxièmement, une heuristique bien paramétrée est développée et sert par le guidage d'un algorithme exact branch-and-bound. Cet algorithme utilise aussi une nouvelle réduction d'espace.
— Troisièmement, le cas particulier du problème sur trois permutations est investigué. Une réduction d'espace de recherche basée sur les graphes est proposée pour ce cas, suivi d'une borne inférieure très stricte. Deux conjectures sont émises et font le lien entre ce cas et le problème du 3-Hitting Set.
— Finalement, une généralisation du problème est proposée et permet d'étendre nos travaux de réduction d'espace de recherche à l'agrégation de classements.

Mots clés : méthode de Kemeny-Young, médiane de permutations, règle de Kemeny, distance de Kendall-τ, agrégation de classements, consensus de classements, votes préférentiels, système de vote, classements, permutations, optimisation, algorithmes

ABSTRACT

A permutation is a list that orders objects or candidates with a preference function or a criterion. Some examples include results from a search engine on the internet, athlete rankings, lists of genes related to a disease given by prediction methods or simply the preference of activities for the next weekend. One might be interested to aggregate a set of permutations to get a consensus permutation. This problem is well known in political science and many methods exists that can aggregate permutations, each one having its mathematical properties. Among those methods, the Kemeny-Young method, also known as the median of permutations, finds a consensus that minimise the sum of distances between that consensus and the set of permutations. This method holds many desirable properties. On the other end, this method is difficult to calculate, thus opening the way for research works. A generalization of this problem considers rankings containing ties between the ranked objects and rankings that might be incomplete by considering only a subset of objects. In this thesis, we study the Kemeny-Young method under different aspects :

— Firstly, a search space reduction technique is proposed. It improves the time complexity of exact algorithms for the problem.

— Secondly, a well parameterized heuristic is developed and is used as guidance in a branch-and-bound exact algorithm. This algorithm also uses a new search space reduction technique.

— Thirdly, the special case of the problem on three permutations is investigated. A search space reduction technique based on graphs is presented for this case, followed by a very tight lower bound. Two conjectures are stated and are linking this case with the 3-Hitting Set problem.

— Finally, a generalization of the problem is proposed and allows us to extend our work on search space reduction techniques to the rank aggregation problem.

Key words : Kemeny-Young method, median of permutations, Kemeny's rule, Kendall-τ distance, rank aggregation, ranking's consensus, preferential votes, voting system, rankings, permutations, optimization, algorithms

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET DES NOTATIONS

$\mathcal{A}$	Ensemble de permutations de départ
$\mathcal{A}^r$	Ensemble de permutations renversées
$a \prec b$	Élément a préféré à l'élément b
$a \equiv b$	Élément a à égalité avec l'élément b
$d_{KT}(\pi,\sigma)$	Distance de Kendall-τ
$E(\mathcal{R})$	Matrice égalité de l'ensemble de classements $\mathcal{R}$
$K(\pi,\mathcal{A})$	Score de Kemeny
$K(R,\mathcal{R})$	Score de Kemeny généralisé
$K^{(p)}(R,\mathcal{R})$	Score de Kemeny généralisé avec pénalité p spécifiée
m	Nombre de permutations de départ (cardinalité de $\mathcal{A}$)
$M(\mathcal{A})$	Ensemble des permutations médianes
n	Taille des permutations
$[n]$	Ensembles des nombres entiers $\{1,2,...,n\}$
$L(\mathcal{A})$	Matrice gauche de l'ensemble de permutations $\mathcal{A}$
$L(\mathcal{R})$	Matrice gauche de l'ensemble de classements $\mathcal{R}$
p	Pénalité de briser une pénalité dans un classement
π, σ, γ, μ	Permutations
π^{-1}	Permutation inverse
π^r	Permutation renversée
π^*	Permutation médiane
R	Classement
$\mathcal{R}$	Ensemble de classements de départ
$R(\mathcal{A})$	Matrice droite de l'ensemble de permutations $\mathcal{A}$
$R(\mathcal{R})$	Matrice droite de l'ensemble de classements $\mathcal{R}$
$Rank_n$	Espace des classements
S_n	Espace des permutations

LISTE DES ABRÉVIATIONS

3HS	3-Hitting Set Problem
FAS	Feedback Arc Set Problem
FPT	Fixed Parameter Tractability
GLPK	Gnu Linear Programming Kit
ILP	Integer Linear Programming
IP	Integer Programming
LOP	Linear Ordering Problem
LP	Linear Programming
LS	Local Search
M3P	Median of 3 Permutations
MOT	Major Order Theorem
PTAS	Polynomial Time Scheme Approximation
SA	Simulated Annealing
VNS	Variable Neighborhood Search
wFAS	weighted Feedback Arc Set Problem

DÉDICACE

À la science.

REMERCIEMENTS

Premièrement, j'aimerais remercier le Fonds de recherche du Québec en Nature et Technologies (FRQNT) qui m'a offert la bourse de doctorat. Cette bourse m'a permis de me consacrer à la recherche en vivant confortablement.

J'aimerais remercier les membres du jury : Pierre L'Écuyer, président, Mathieu Blanchette, examinateur externe, Nadia El-Mabrouk, membre, et Martin Bilodeau, le représentant du doyen pour avoir accepté d'être sur le jury de cette thèse.

J'aimerais aussi remercier l'organisme Mitacs qui m'a offert la bourse de mobilité Globalink. Cette bourse m'a permis de faire un stage de recherche au Laboratoire de Recherche en Informatique (LRI) à l'Université Paris-Sud, en me permettant, du coup, de passer une demie année sur le Vieux Continent. C'est dans ce merveilleux stage, que j'ai eu la chance de travailler avec Pierre Andrieu, un brillant mathématicien et étudiant au doctorat en informatique. Merci à Sarah Cohen-Boulakia qui m'a si bien accueilli au LRI et qui m'a donné des belles opportunités. Merci aussi à tous ceux avec qui j'ai travaillé au courant de ce stage : Alain Denise, Adeline Pierrot, Christelle Rovetta et Bryan Brancotte. Merci à Piotr et Anna Tucholka, qui ne m'ont pas seulement hébergé avec ma copine chez eux en France, mais qui nous on accueillis comme une vraie famille.

Je tiens à remercier les membres du Laboratoire de Biologie Informatique et Théorique (LBIT) que j'ai côtoyé au cours de mes trois années de doctorat : Emmanuel Noutahi, Nadia El-Mabrouk, Manuel Lafond, Charles Desharnais, Anna Thibert, Samuel Briand, Miguel Sautié et Marie Gasparoux. Merci aussi à Marie Pageau, la responsable des laboratoires en bio-informatique de l'UdeM, qui m'a aidé maintes fois dans mon enseignement des démonstrations des cours de bio-informatique.

Merci à Adam Desjardins, enseignant d'informatique au secondaire qui m'a introduit à l'informatique et a su cultiver ma curiosité dans ce domaine. Merci à l'Association Mathématique du Québec (AMQ) qui, en m'invitant dans ces camps d'été, m'a donné goût au mathématiques. On peut maintenant bien voir les racines de mes études en informatique et mathématiques.

Merci à mes amis, avec qui j'ai passé de très beaux moments et m'ont gardé un pied sur terre (ou sur un fil !), alors que ma tête était souvent en "mode recherche".

Je tiens à remercier chaleureusement ma mère, Anna Klepacka, qui m'a offert un soutien bien apprécié au cours de mes études, et mon amour, Nicole Burke, qui m'a tellement encouragé et qui m'a accompagné dans mes aventures académiques menant à différents coins du monde.

Finalement, un énorme merci à Sylvie Hamel, ma directrice de recherche qui m'a bien guidé et suivi au cours de ce doctorat, et qui m'a aussi laissé une grande autonomie dans ma recherche. Elle m'a introduite au merveilleux domaine de la bio-informatique et m'a offert des magnifiques opportunités académiques. Je me sens privilégié d'avoir travaillé avec Sylvie. Ce doctorat fut une très belle aventure grâce à elle.

INTRODUCTION

Le problème de la médiane d'un ensemble $\mathcal{A}$ de m permutations des éléments $\{1,2,\ldots n\}$ [76] sous la distance de Kendall-τ [78] est un problème d'optimisation combinatoire où l'on veut trouver la permutation appelée médiane qui minimise la somme des distances de celle-ci aux m permutations de l'ensemble de départ $\mathcal{A}$. La distance de Kendall-τ entre 2 permutations compte le nombre de paires d'éléments dont l'ordre relatif est en désaccord.

En science politique, ce problème est aussi connu sous le nom de "méthode de Kemeny-Young" [140]. Dans ce contexte, on appelle "consensus" une permutation médiane. Le problème est alors présenté comme un système électoral où m électeurs vont individuellement ordonner une liste de n candidats selon leurs préférences. En comparaison à d'autres systèmes électoraux préférentiels, la méthode de Kemeny-Young possède plusieurs propriétés intéressantes.

Le problème qui est trivial à résoudre pour $m = 1$ et $m = 2$, a été démontré NP-Difficile quand $m \geq 4$ avec m pair [56], et récemment, il a été suggéré être NP-Difficile pour $m \geq 7$, m impair [11]. Il n'existe donc pas de façon facile de résoudre ce problème et plusieurs approches ont été explorées pour s'attaquer à celui-ci. La complexité pour les cas $m = 3$ et $m = 5$ demeure encore inconnue.

Une version généralisée du problème traite du cas où des éléments ou candidats peuvent se trouver à égalité alors que d'autres éléments peuvent être simplement absents. On parle alors du consensus de classements.

Le problème de la médiane de permutations trouve des applications dans plusieurs domaines dont la bio-informatique, les sciences médicales, les sciences sociales, les sciences environnementales, l'informatique, etc.

Les premiers travaux sur la médiane de permutations ont eu lieu à la fin du 20e siècle et ont eu trait au propriétés mathématiques de cette méthode [138, 139, 140]. Au début du 21e siècle, ce problème est revenu à l'attention des communautés scientifiques et de nombreux travaux ont été réalisés depuis. Ceux-ci étant principalement axé sur les méthodes heuristiques [2, 3, 4, 6, 8, 12, 25, 39, 51, 56, 49, 61, 92, 117, 135], la résolution

exacte [44, 48, 51, 92], la complexité [14, 39, 56, 72] et les algorithmes à paramètres fixes [18, 19, 20, 21, 22, 63, 75, 104, 123]. D'autres axes de recherche secondaires ont aussi été investigués tels les bornes inférieures [44, 51], les méthodes de réductions d'espace de recherche [18, 26, 30, 40, 56, 96, 133] et un schéma d'approximation en temps polynomial [80]. Les travaux se rapportant aux consensus de classements sont encore plus récents et bien moins nombreux [2, 19, 21, 59].

Dans le plus récent travail majeur de comparaison d'heuristiques et d'algorithmes pour ce problème [6], les auteurs ont remarqué qu'il y a très peu de travaux qui se sont intéressés aux méthodes de réduction d'espace. Nous avons alors investigué plus en détails cette approche, obtenant des résultats pour la médiane de permutations ainsi que pour le consensus de classements. Nos travaux ont aussi touché aux bornes inférieures, elles aussi peu étudiées pour ce problème. Nous avons développé un algorithme raffiné de branch-and-bound très compétitif. Alors qu'un éventail d'heuristiques ont été proposées dans la littérature, nous avons travaillé sur des heuristiques donnant des résultats très proches de l'optimal pour des problèmes de taille moyenne, une approche encore une fois moins explorée dans la littérature. Les travaux dans cette thèse sont donc principalement axés sur les méthodes de réduction d'espace, les bornes inférieures, les approches exactes et les heuristiques.

Il est à noter que ce travail s'inscrit dans une suite de travaux dont fait partie mon mémoire de maîtrise. Les deux contributions majeures de ce mémoire ont été l'étude de classes spéciales du problème et la réduction d'espace de recherche. Ce mémoire contenait aussi des idées embryonnaires d'heuristiques et d'algorithmes branch-and-bound.

Cette thèse se divise en six chapitres, énumérés ci-dessous, qui concernent ces trois volets : la définition et le contexte du problème, les travaux antérieurs pertinents et les applications, puis le travail complété dans le cadre du doctorat. La thèse est encadrée par une introduction et une conclusion. Elle se base sur mon rapport prédoctoral [95] présenté en août 2017 au département d'informatique (DIRO) de l'Université de Montréal. Plus précisément, nous avons la division suivante :

Le chapitre 1 introduit le problème et son contexte. On y introduit les définitions nécessaires à la compréhension de la thèse et on formalise mathématiquement le problème. Le chapitre introduit aussi le problème généralisé du consensus de classements.

Le chapitre 2 fait le tour des travaux qui ont été réalisés sur ce problème dans une revue de littérature. On y aborde la complexité, les méthodes heuristiques, un schéma d'approximation en temps polynomial, les algorithmes à paramètres fixes, la réduction

d'espace, les bornes inférieures, les approches de résolution exacte et les travaux de comparaison. On y discute aussi brièvement des données utilisées dans les travaux sur le sujet et de leur format.

Le chapitre 3 énumère plusieurs exemples d'applications dans divers domaines scientifiques et sociaux. On y fait aussi une brève remarque sur le vote stratégique.

Le chapitre 4 est la suite des travaux de maîtrise et consiste en un article journal publié dans le Journal of Discrete Applied Mathematics [**98**], c'est la version étendue de l'article présenté à CALDAM2016 [**96**] (travail de maîtrise).

Le chapitre 5 présente des travaux sur la médiane de permutations qui ont trait à la réduction d'espace, aux heuristiques et aux approches exacte. Le chapitre débute avec un article qui a été présenté à la conférence IWOCA2017 (Internationnal Workshop On Combinatorial Algorithms), en juillet 2017 [**97**]. Le chapitre clôture avec une discussion sur la version journal de cet article, qui vient tout juste d'être acceptée dans le Journal of Discrete Algorithms [**99**] en octobre 2018, et qui se trouve en annexe de la thèse.

Le chapitre 6 présente un travail important effectué durant un stage de recherche Mitacs au Laboratoire de Recherche en Informatique (Paris-Saclay, France) : une étude approfondie du problème de la médiane de trois permutations. Cet article a été présenté à la conférence IWOCA2018, en juillet 2018 [**100**].

Le chapitre 7 offre une généralisation du problème aux ensembles de classements avec ou sans égalités. Y est présenté en premier temps, un article en préparation qui traite des propriétés mathématiques des agrégations de classements. En deuxième temps, une petite contribution à un second article en préparation est brièvement discutée.

Chapitre 1

DÉFINITIONS ET NOTATIONS

De façon informelle, le problème de la médiane de permutations consiste à trouver la médiane (un consensus) d'un ensemble de votes où des candidats sont ordonnés dans une liste de préférence (permutations). Le critère de consensus utilisé vise à garder l'ordre relatif des paires d'éléments au plus possible similaire à celui des votes. Un exemple est donné à la Figure 1.1 puis à l'exemple 1.0.1.

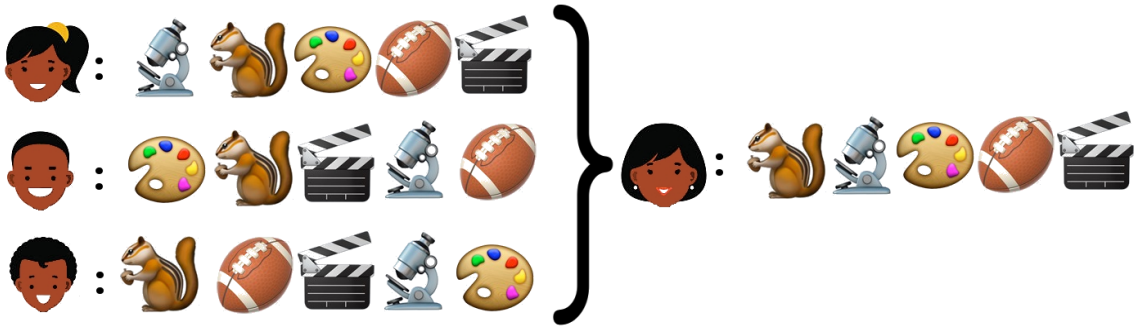


FIGURE 1.1. Exemple d'introduction : Dans une famille, les parents proposent cinq activités pour la fin de semaine : faire un projet de peinture, jouer au ballon, aller au cinéma, visiter le centre des sciences et aller au parc nature. Les trois enfants ordonnent les activités proposées par les parents en fonction de leurs préférences personnelles. La mère, qui fait ses études en mathématiques, utilise la méthode de Kemeny-Young pour trouver un consensus. L'exemple 1.0.1 présente la modélisation mathématique de cette figure.

Exemple 1.0.1. *La modélisation du problème de la Figure 1.1 avec des permutations est la suivant : chaque activité se voit associer un entier différent de 1 à 5. Ici, nous avons : 1) faire un projet de peinture, 2) jouer au ballon, 3) aller au cinéma, 4) visiter le centre des sciences et 5) aller au parc nature. Nous obtenons donc les permutations suivantes :*

$$\left.\begin{aligned}\pi_1 &= [4,5,1,2,3]\\ \pi_2 &= [1,5,3,4,2]\\ \pi_3 &= [5,2,3,4,1]\end{aligned}\right\} \quad \pi^* = [5,4,1,2,3]$$

Le problème a été prouvé NP-Difficile dans [56]. Il est alors intéressant d'étudier des classes d'instances où la résolution est plus facile, des propriétés qui nous donnent des indices sur la forme du consensus et des heuristiques qui peuvent nous donner des approximations du consensus. Mais tout d'abord, nous allons commencer en formalisant le problème dans ce qui suit.

1.1. Définitions et notations sur les permutations

1.1.1. Permutations, ensembles de permutations et espace

Définition 1.1.1. *Une* **permutation** π *de n éléments est une bijection de l'ensemble $\{1,2,\ldots,n\}$ sur lui-même.*

Définition 1.1.2. *La* **composition** *de permutations $\gamma = \sigma \circ \pi$ est définie comme : $\gamma_i = \sigma_{\pi_i}$. L'***inverse** π^{-1} *d'une permutation π est la permutation telle que $\pi^{-1} \circ \pi = I$ où I dénote la permutation identité $I = [1,2,3,...,n]$.*

On dénote une permutation π de $\{1,2,\ldots,n\}$ par $\pi = \pi_1\pi_2\ldots\pi_n$, où π_j dénote l'entier à la position j et π_i^{-1} dénote la position de l'élément i. Un exemple de permutation se trouve à l'Exemple 1.1.1.

Définition 1.1.3. *L'***ordre entre deux éléments** *(ou la relation de précédence pour une paire d'éléments) d'une permutation est définit comme suit : $a \prec_\pi b$ signifie que a est préféré à b dans la permutation π (ou que a est à gauche de b, ou que a est avant b, mathématiquement $\pi_a^{-1} < \pi_b^{-1}$) alors que $b \prec_\pi a$ signifie que b est préféré à a. Comme une permutation est un ordre total, chaque paire d'éléments a un ordre bien défini.*

Notez que dans le cas où il n'y a pas d'ambiguïté, la notation $a \prec_\pi b$ est parfois simplifiée à $a \prec b$. La notation $a \succ b$ est équivalente à $b \prec a$.

Exemple 1.1.1. *La permutation $\pi = [4,5,1,2,3]$ est une permutation de taille $n = 5$. À la position 4 se trouve l'élément 2, noté $\pi_4 = 2$. La position de l'élément 3 est $\pi_3^{-1} = 5$. Dans π, l'élément 2 précède l'élément 3 et donc on note $2 \prec_\pi 3$ (ou $2 \prec 3$).*

Définition 1.1.4. *Une* **relation d'ordre**, *notée* $\leq$, *sur un ensemble* E *est un relation binaire qui vérifie la transitivité* ($x \leq y, y \leq z \implies x \leq z,\ x,y,z \in E$), *la réflexivité* ($x \leq x,\ x \in E$) *et l'antisymétrie* ($x \leq y, y \leq x \implies x = y,\ x,y \in E$). *Un* **ordre total** *est une relation d'ordre dans laquelle* $x \leq y$ *ou* $y \leq x,\ \forall x \neq y \in E$. *Si une relation n'est pas un ordre total, on dit que c'est un* **ordre partiel**.

Une permutation peut alors être vue comme un ordre total des éléments $\{1,2,\ldots,n\}$ avec la relation de précédence $\prec$: toutes les paires d'éléments sont comparables puis la transitivité, la réflexivité et l'antisymétrie sont vérifiées. Cette façon de voir va devenir utile plus tard (Section 1.2) quand on va considérer les classements avec des éléments à égalité, dans ce cas on parlera d'ordre partiel.

Définition 1.1.5. *Le* **groupe symétrique** S_n *est l'ensemble contenant toutes les permutations de taille* n *muni de la composition (Définition 1.1.2). L'élément neutre est la permutation identité* $I = [1,2,3,...,n]$.

Le groupe symétrique sera notre espace de recherche (voir Figure 1.2 pour un exemple visuel).

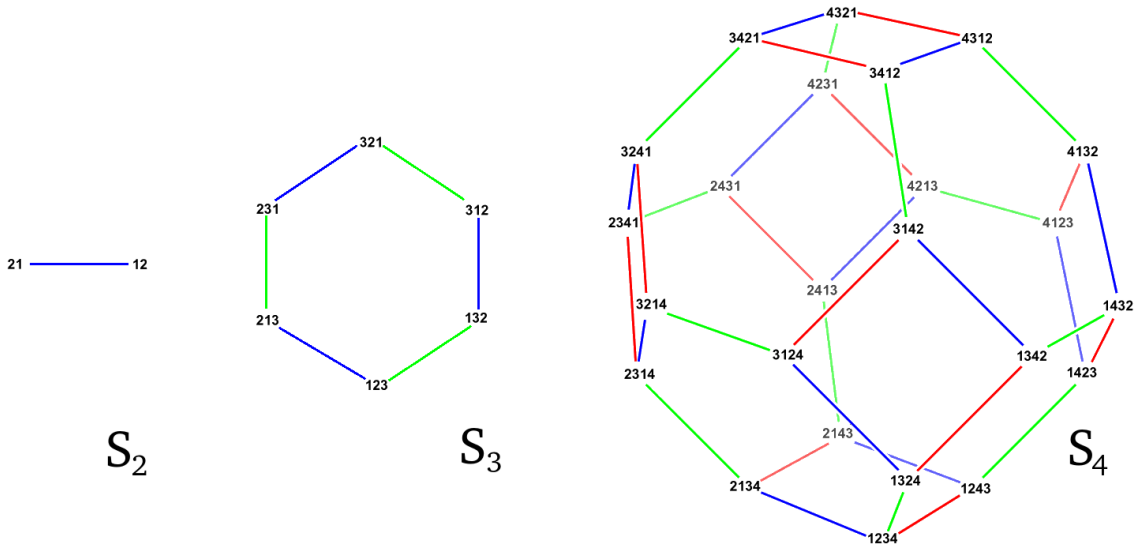


FIGURE 1.2. Le groupe symétrique, exemples pour S_2, S_3 et S_4. Ici, une arête est créée entre deux permutations si elles ne diffèrent que d'une transposition d'éléments adjacents. Ces polytopes ainsi créés sont appelés des permutahedrons. La couleur des arêtes indique la position des transpositions : bleu pour une transposition des éléments en position 1 et 2, vert pour les positions 2 et 3 puis rouge pour les positions 3 et 4.

On note l'ensemble des permutations de départ de notre problème par $\mathcal{A}$. Cet ensemble de permutations caractérise une instance du problème. Ce sont les votes sur les candidats. On note m le nombre $\#\mathcal{A}$ de permutations dans l'ensemble $\mathcal{A}$. Toutes les permutations sont de taille identique notée n. Ces deux premières caractéristiques (m et n) vont déjà être utiles pour discriminer la difficulté du problème. On peut aussi permettre la répétition de permutations dans l'ensemble de départ et dans ce cas $\mathcal{A}$ devient un multiensemble ou une collection de permutations.

Pour caractériser un ensemble (ou une collection) de permutations $\mathcal{A}$ on définit deux matrices : $L(\mathcal{A})$ la **matrice gauche** et $R(\mathcal{A})$ la **matrice droite** qui comptent le nombre de permutations dans lesquelles un élément i est à gauche de j ($i \prec j$) ou l'inverse. $L(\mathcal{A})$ peut être aussi appelée **matrice de précédence** [6].

Définition 1.1.6. *$L_{ij}(\mathcal{A})$ (resp. $R_{ij}(\mathcal{A})$) dénote le nombre de permutations de l'ensemble $\mathcal{A}$ où l'élément i est à gauche de l'élément j, $i \prec j$ (resp. i à droite de j, $j \prec i$), $1 \leq i \neq j \leq n$:*

$$L_{ij}(\mathcal{A}) = \#\{\pi \in \mathcal{A} \mid \pi_i^{-1} < \pi_j^{-1}\}, \text{ et}$$
$$R_{ij}(\mathcal{A}) = \#\{\pi \in \mathcal{A} \mid \pi_i^{-1} > \pi_j^{-1}\}.$$

On note $i \prec_{\mathcal{A}} j$ si $L_{ij}(\mathcal{A}) > R_{ij}(\mathcal{A})$. Par définition on a que : $L_{ij}(\mathcal{A}) + R_{ij}(\mathcal{A}) = m = \#\mathcal{A}$ et $L_{ij}(\mathcal{A}) = R_{ji}(\mathcal{A})$ (i.e. $L^t(\mathcal{A}) = R(\mathcal{A})$ et $R^t(\mathcal{A}) = L(\mathcal{A})$). L'exemple 1.1.2 donne les matrices gauches et droites de l'exemple 1.0.1.

Exemple 1.1.2. *Les matrices droite $R(\mathcal{A})$ et gauche $L(\mathcal{A})$ pour l'ensemble $\mathcal{A}$ de l'exemple 1.0.1 d'introduction.*

$$R = \begin{bmatrix} 0 & 1 & 1 & 2 & 2 \\ 2 & 0 & 1 & 2 & 3 \\ 2 & 2 & 0 & 1 & 3 \\ 1 & 1 & 2 & 0 & 2 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix} \quad L = \begin{bmatrix} 0 & 2 & 2 & 1 & 1 \\ 1 & 0 & 2 & 2 & 0 \\ 1 & 1 & 0 & 2 & 0 \\ 2 & 2 & 1 & 0 & 1 \\ 2 & 3 & 3 & 2 & 0 \end{bmatrix}$$

Une autre façon de caractériser un ensemble (ou une collection) de permutations est le graphe de majorité qui permet de visualiser les préférences entres les éléments.

Définition 1.1.7. *Soit $\mathcal{A}$ un ensemble de permutations. Le* **graphe de majorité** *$G_{\mathcal{A}} = (E,V)$ est un graphe de tournoi pondéré tel que $V = \{1,...,n\}$, $E = \{(i,j) | L_{ij}(\mathcal{A}) > R_{ij}(\mathcal{A})\}$ et $w(i,j) = L_{ij}(\mathcal{A}) - R_{ij}(\mathcal{A})$.*

Un exemple de graphe de majorité (associé à l'ensemble $\mathcal{A}$ de l'Exemple 1.0.1) se trouve à la Figure 1.3.

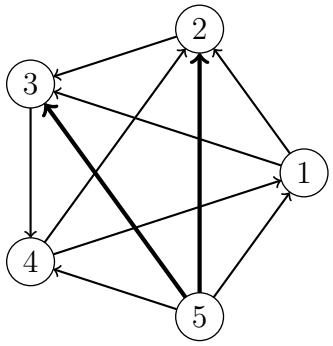


FIGURE 1.3. Le graphe de majorité pour l'ensemble $\mathcal{A}$ = {[4,5,1,2,3], [1,5,3,4,2], [5,2,3,4,1]}. Les arcs en gras ont un poids de 3 alors que les autres ont un poids de 1.

Définition 1.1.8. *Soit $\mathcal{A}$ un ensemble de permutations. L'***ordre majoritaire** *d'une paire d'éléments (i,j), $1 \leq i \neq j \leq n$ est l'ordre relatif $i \prec j$ ou $j \prec i$ qui apparaît dans la majorité des permutations de l'ensemble de départ $\mathcal{A}$, et inversement pour l'ordre minoritaire. Si $i \prec j$ et $j \prec i$ apparaissent dans un nombre égal de permutations, alors on dit qu'il n'y a pas d'ordre majoritaire. Si $i \prec j$ est l'ordre majoritaire pour $\mathcal{A}$ alors on note $i \prec_{\mathcal{A}} j$.*

1.1.2. Distance de Kendall-τ, score de Kemeny et problème de la médiane

Avec l'objectif de trouver un consensus d'un ensemble $\mathcal{A}$ de permutations de taille n, nous allons introduire dans la sous-section suivante les notions de distance de Kendall-τ entre les permutations et de score de Kemeny d'une permutation. Ce score permet de définir le critère qui indique si une permutation est une médiane de $\mathcal{A}$. Le problème de la médiane consiste à retourner une ou plusieurs médianes d'un ensemble de permutations.

Définition 1.1.9. *La* **distance de Kendall-τ** *entre deux permutations compte le nombre de paires d'éléments qui sont en désaccord par rapport à leur ordre :*

$$d_{KT}(\pi,\sigma) = \#\{(i,j) | i < j \wedge (\pi_i^{-1} < \pi_j^{-1} \wedge \sigma_i^{-1} > \sigma_j^{-1}) \vee (\pi_i^{-1} > \pi_j^{-1} \wedge \sigma_i^{-1} < \sigma_j^{-1}).\}$$

La distance maximale de Kendall-τ entre deux permutations est de $\frac{n(n-1)}{2}$ et représente le cas où toutes les paires d'éléments sont en désaccord. C'est le cas où $\pi = \pi_1\pi_2...\pi_n$ et $\sigma = \pi_n\pi_{n-1}...\pi_1$, i.e. les permutations π et π^r. La distance de Kendall-τ se calcule naïvement en $O(n^2)$ en comparant toutes les paires et en $O(nlogn)$ en utilisant une version

modifiée de l'algorithme de tri fusion. Il suffit de réétiqueter les éléments pour que l'une des permutations devienne l'identité et trier la seconde avec le tri fusion en comptant le nombre de "sauts" des éléments de la 2e liste lors de la fusion. Il est intéressant de mentionner que la distance de Kendall-τ est équivalente à la distance du tri à bulles (*Bubble Sort*) qui compte le nombre de transpositions d'éléments adjacents nécessaire pour passer d'une permutation à une autre (voir exemples 1.1.3 et 1.1.4).

Exemple 1.1.3. *Distance de Kendall-τ entre deux permutations π et σ. Les paires en rouges représentent les désaccords et leur nombre (5) est la distance de Kendall-τ :*

$$\pi = [4,5,1,2,3]$$
$$\sigma = [1,5,3,4,2]$$

(1,2)	**(2,3)**	**(3,4)**	**(4,5)**
(1,3)	*(2,4)*	*(3,5)*	
(1,4)	*(2,5)*		
(1,5)			

Exemple 1.1.4. *Distance de Bubble Sort entre deux permutations π et σ. On peut observer les 5 transpositions d'éléments adjacents pour passer de π à σ :*

$$\begin{aligned}
\pi = \quad & [4,5,1,2,3] \\
& [\mathbf{5},\mathbf{4},1,2,3] \\
& [5,\mathbf{1},\mathbf{4},2,3] \\
& [\mathbf{1},\mathbf{5},4,2,3] \\
& [1,5,4,\mathbf{3},\mathbf{2}] \\
\sigma = \quad & [1,5,\mathbf{3},\mathbf{4},2]
\end{aligned}$$

La distance de Kendall-τ peut aussi être définie ([**12**, **48**, **58**]) sous une forme matricielle [**77**]. Une permutation est alors représentée comme une matrice α dans laquelle $\alpha_{ij} = 1$ si $i \prec j$, $\alpha_{ij} = -1$ si $j \prec i$ et $\alpha_{ij} = 0$ si $i = j$. La distance de Kendall-τ est alors notée :

$$d(\pi,\sigma) = \frac{1}{2}\sum_{i=1}^{n-1}\sum_{j>i}^{n}|\alpha_{ij} - \beta_{ij}|,$$

où α et β représentent respectivement les permutations π et σ. Cette forme permet une généralisation facile aux classements incomplets et avec égalités.

On généralise la distance de Kendall-τ entre 2 permutations à un score entre une permutation et un ensemble de permutations de la façon suivante :

Définition 1.1.10. *Le* **score de Kemeny** *est définit entre une permutation et un ensemble de permutations, toutes sur le même ensemble d'éléments de la façon suivante : Soit une permutation $\pi \in S_n$ et un ensemble de permutations $\mathcal{A} \subseteq S_n$ alors :*

$$K(\pi, \mathcal{A}) = \sum_{\sigma \in \mathcal{A}} d_{KT}(\pi, \sigma).$$

Une façon équivalente de calculer le score de Kemeny repose sur les matrices $R(\mathcal{A})$ et $L(\mathcal{A})$:

$$K(\pi, \mathcal{A}) = \sum_{\substack{i \neq j \in \{1,\dots,n\} \\ i \prec_\pi j}} R_{ij}(\mathcal{A}).$$

Définition 1.1.11. *Le* **problème de la médiane de permutations**, *aussi nommé* **la méthode de Kemeny-Young**, *vise à trouver la ou les permutations de S_n qui minimisent la distance entre elles et l'ensemble $\mathcal{A}$: Soit $\mathcal{A} \subseteq S_n$ un ensemble de permutations, trouver l'ensemble $M(\mathcal{A})$ des permutations médianes (aussi appelée* **consensus de Kemeny***) $\pi^* \in S_n$ tel que*

$$K(\pi^*, \mathcal{A}) \leq K(\pi, \mathcal{A}), \forall \pi \in S_n.$$

Notez que la taille de l'ensemble des permutations médianes $M(\mathcal{A})$ peut varier énormément d'un ensemble $\mathcal{A} \subseteq S_n$ à un autre.

Définition 1.1.12. *Le* **score optimal de Kemeny** *(parfois appelé simplement "Kemeny score") $K(\pi^*,\mathcal{A})$ est le score de Kemeny d'une médiane :*

$$K(\pi^*, \mathcal{A}) = min_{\pi \in S_n}\{K(\pi, \mathcal{A})\}$$

La Figure 1.4 illustre la médiane de l'ensemble de permutations de l'Exemple 1.0.1.

Dans un problème connexe très proche, on peut être intéressé à ne pas avoir nécessairement toutes les médianes mais qu'un sous-ensemble de celles-ci (comme dans le chapitre 5) ou dans une autre variante, trouver seulement le score optimal de Kemeny.

Finalement, les problèmes de décision [72] reliés au problème de la médiane de permutation s'énoncent comme ci :

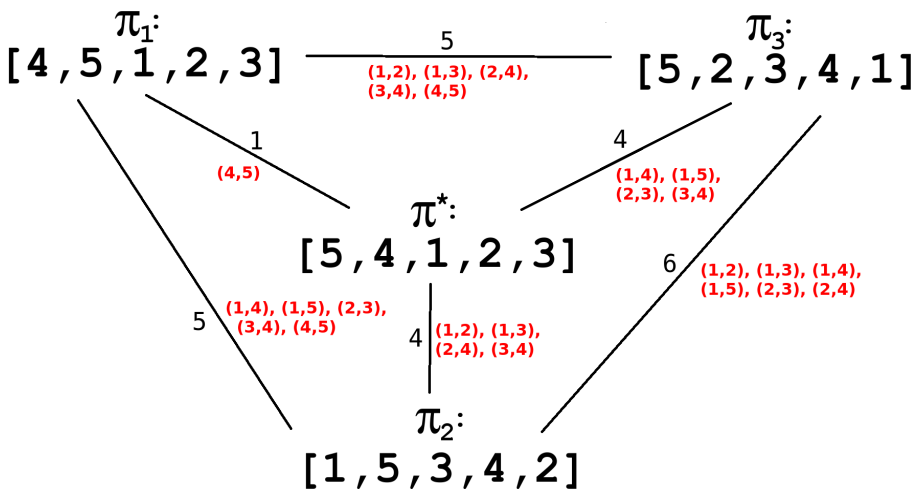


FIGURE 1.4. Médiane de l'ensemble présenté à l'exemple 1.0.1. L'ensemble $\mathcal{A}$ contient les permutations π_1 = [4,5,1,2,3],π_2 = [1,5,3,4,2] et π_3 = [5,2,3,4,1]. L'unique permutation médiane est π^* = [5,4,1,2,3]. Les distances de Kendall-τ entre les permutations sont indiquées en noir et les paires d'éléments en désaccords sont indiquées en rouge. Le score de Kemeny est de 1 + 4 + 4 = 9 dans ce cas.

Problème de décision 1.1.1. Kemeny Score

Étant donné un ensemble de m permutations $\mathcal{A} \subseteq S_n$ et un entier k, existe-t'il une permutation π^* tel que $K(\pi^*, \mathcal{A}) \leq k$.

Problème de décision 1.1.2. Kemeny Winner

Étant donné un ensemble de m permutations $\mathcal{A} \subseteq S_n$ et un élément c, $1 \leq c \leq n$, existe-t'il une permutation médiane π^* dans laquelle c est le premier élément, i.e. $c = \pi^*_1$.

Problème de décision 1.1.3. Kemeny Ranking

Étant donné un ensemble de m permutations $\mathcal{A} \subseteq S_n$ et une paire d'éléments c et d, $1 \leq c \neq d \leq n$, est-ce que c est préféré (ou égalise) à d dans toutes les permutations médianes.

1.1.3. Modélisation équivalente

On peut aussi modéliser le problème de la médiane d'un ensemble de permutations $\mathcal{A} \subseteq S_n$ en programmation en nombres entiers (Integer Linear Programming ou ILP) en

utilisant la modélisation standard du problème d'ordonnancement linéaire (Linear Ordering Problem ou LOP) :

$$minimiser : \sum_{i \neq j,\ i,j \in \{1,...,n\}} R_{ij}(\mathcal{A}) x_{ij} \tag{1.1.1}$$

sujet aux contraintes :

$$x_{ij} + x_{ji} = 1, \quad i \neq j,\ i,j \in \{1,...,n\} \tag{1.1.2}$$

$$x_{ij} + x_{jk} + x_{ki} \geq 1, \quad i \neq j \neq k,\ i,j,k \in \{1,...,n\} \tag{1.1.3}$$

$$x_{ij} \in \{0,1\} \quad i \neq j,\ i,j \in \{1,...,n\} \tag{1.1.4}$$

Les variables x_{ij} définissent l'ordre entre les éléments i et j : si $x_{ij} = 1$ alors i précède j ($i \prec j$) dans la solution. Dans la fonction de coût (1.1.1), $R_{ij}(\mathcal{A})$ indique le nombre de permutations dans $\mathcal{A}$ dans lesquelles i est à droite de j, alors si i précède j dans la solution ($x_{ij} = 1$), on doit additionner $R_{ij}(\mathcal{A})$ permutations en désaccords avec cette paire dans le score de Kemeny.

La contrainte (1.1.2) $x_{ij} + x_{ji} = 1$ force à avoir un ordre soit $i \prec j$ soit $j \prec i$ ou entre les deux. La contrainte (1.1.3) $x_{ij} + x_{jk} + x_{ki} \geq 1$ force la transitivité de la relation de précédence : si $i \prec j$ et $j \prec k$ alors $i \prec k$. Cette contrainte empêche aussi les cycles de longueur 3, car $x_{ij} = 1$, $x_{jk} = 1$ et $x_{ki} = 1$ donnerait $x_{ji} = 0$, $x_{kj} = 0$ et $x_{ik} = 0$. Il est trivial de montrer que dans un graphe de tournoi, s'il y a un cycle de longueur $k > 3$, il y a alors un cycle de longueur 3. Il suffit de considérer trois sommets consécutifs c_{i-1}, c_i, c_{i+1} du cycle $C = (c_1,...,c_k)$: soit ils forment un 3-cycle (c_{i-1}, c_i, c_{i+1}), soit on obtient un cycle de longueur $k-1$: $C' = (c_1,...c_{i-1}, c_{i+1}, ..., c_k)$ et on considère récursivement trois nouveaux sommets consécutifs de C' jusqu'à éventuellement obtenir un 3-cycle. En empêchant tout cycle de longueur 3, la contrainte oblige la solution à être acyclique.

La contrainte (1.1.4) $x_{ij} \in \{0,1\}$ oblige les variables d'ordre à être bornées et entières, alors on a soit $i \prec j$ ou $j \prec i$.

La résolution de ce problème ILP donne le score optimal de Kemeny et des assignations de variables que l'on peut convertir en une permutation médiane pour l'ensemble $\mathcal{A}$. Différents logiciels commerciaux ou open source sont disponibles pour résoudre cette modélisation tels que CPLEX [1], Gurobi [2], GLPK [3], Xpress [4] et bien d'autres.

1. https://www.ibm.com/ca-en/marketplace/ibm-ilog-cplex
2. http://www.gurobi.com/products/gurobi-optimizer
3. https://www.gnu.org/software/glpk/
4. http://www.fico.com/en/products/fico-xpress-solver

Si on enlève la dernière contrainte (contrainte d'intégralité) pour la remplacer avec une contrainte de bornes $x_{ij} \in [0,1]$, on obtient une borne inférieure intéressante pour le problème par la méthode du simplexe.

Il est évident que la matrice $R(\mathcal{A})$ (ou $L(\mathcal{A})$) donne l'information suffisante pour résoudre le problème de la médiane. Dans ce cas, l'ensemble de permutations $\mathcal{A}$ ne sert qu'à construire ces matrices.

Une modélisation équivalente, retrouvée parfois dans la littérature [**39**], considère la maximisation des accords à la place de minimiser les désaccords. Le problème est alors ainsi formulé :

$$maximiser : \sum_{i \neq j,\ i,j \in \{1,\ldots,n\}} L_{ij}(\mathcal{A}) x_{ij} \tag{1.1.5}$$

sujet aux contraintes :

$$x_{ij} + x_{ji} = 1, \quad i \neq j,\ i,j \in \{1,\ldots,n\} \tag{1.1.6}$$

$$x_{ij} + x_{jk} + x_{ki} \geq 1, \quad i \neq j \neq k,\ i,j,k \in \{1,\ldots,n\} \tag{1.1.7}$$

$$x_{ij} \in \{0,1\} \quad i \neq j,\ i,j \in \{1,\ldots,n\} \tag{1.1.8}$$

Les variables sont les mêmes que pour le modèle de minimisation.

1.2. Introduction aux classements

Dans le problème de la médiane de permutations, on s'intéresse à trouver un consensus entre plusieurs listes (votes) ordonnées de candidats ce qui implique qu'il doit exister un ordre de préférence stricte entre n'importe quels deux candidats pour chacun des votes. Cela n'est pas toujours possible dans la réalité avec des situations dans lesquelles il est impossible de distinguer des candidats entre eux ou qu'ils sont simplement jugés être à égalité (un exemple est donné dans le Tableau 1. I). C'est pourquoi on peut s'intéresser à une généralisation du problème : celle de considérer des classements où la relation d'égalité ($i = j$) est permise entre deux candidats dans les votes. On va utiliser ici le terme "classement" pour désigner un classement pouvant contenir des égalités, par opposition à la permutation qui ordonne les candidats avec un ordre strict ($i \prec j$ ou $j \prec i$, $\forall i,j,\ i \neq j$). De plus, on pourra considérer des ensembles de classements dans lesquels les classements ne couvrent pas nécessairement tous les éléments.

De plus, les classements peuvent contenir des éléments absents (Voir Tableau 1. II).

Restaurants	★★★★★	☆★★★★	☆☆★★★	☆☆☆★★	☆☆☆☆★
A) Chez Septime	•	○	○	○	○
B) Krusty Burger	○	○	○	•	○
C) Joe's Diner	○	○	•	○	○
D) The Shed at Dulwich	•	○	○	○	○
E) Snack-bar chez Raymond	○	○	○	○	•
F) La Ratatouille	○	•	○	○	○
G) Soul Food Cafe	○	○	○	•	○

TABLEAU 1. I. "Quelle note donneriez-vous aux restaurants suivants ?". Exemple de questionnaire sur la qualité de divers restaurants. Ce type de questionnaire est très populaire en sciences sociales. Un remplissage correct de ce questionnaire peut être interprété comme un classement avec égalités i.e. R = [[A,D],[F],[C],[B,G],[E]].

Enjeux politiques	Importance de l'enjeu						
	+++	++	+	+/-	-	- -	- - -
1. environnement	•	○	○	○	○	○	○
2. sécurité	○	○	○	○	○	•	○
3. éducation	•	○	○	○	○	○	○
4. transport	○	○	○	○	•	○	○
5. santé	○	•	○	○	○	○	○
6. culture	○	○	•	○	○	○	○
7. tourisme	○	○	○	○	○	○	○
8. économie	○	○	•	○	○	○	○
9. justice	○	○	•	○	○	○	○

TABLEAU 1. II. "Quelle importance donnez-vous aux enjeux politiques suivants pour la prochaine élection ?" Exemple de questionnaire qui engendre des classements pour des enjeux de société. Ce type de questionnaire est très populaire en sciences sociales. Un remplissage correct de ce questionnaire peut être interprété comme un classement avec égalités i.e. R = [[1,3],[5],[6,8,9],[4],[2]]. Notez qu'on n'utilise pas de "paniers" vides pour les classes d'importance non utilisées (ici +/- et - - -). Notez aussi que l'élément 7 est absent du classement.

Les classements ont aussi leur utilité dans le domaine de la bio-informatique [**30**] [**40**] où plusieurs méthodes de prédictions donnent des classements différents de gènes liées à une certaine maladie en ordre d'importance (Voir l'exemple de la Figure 1.5).

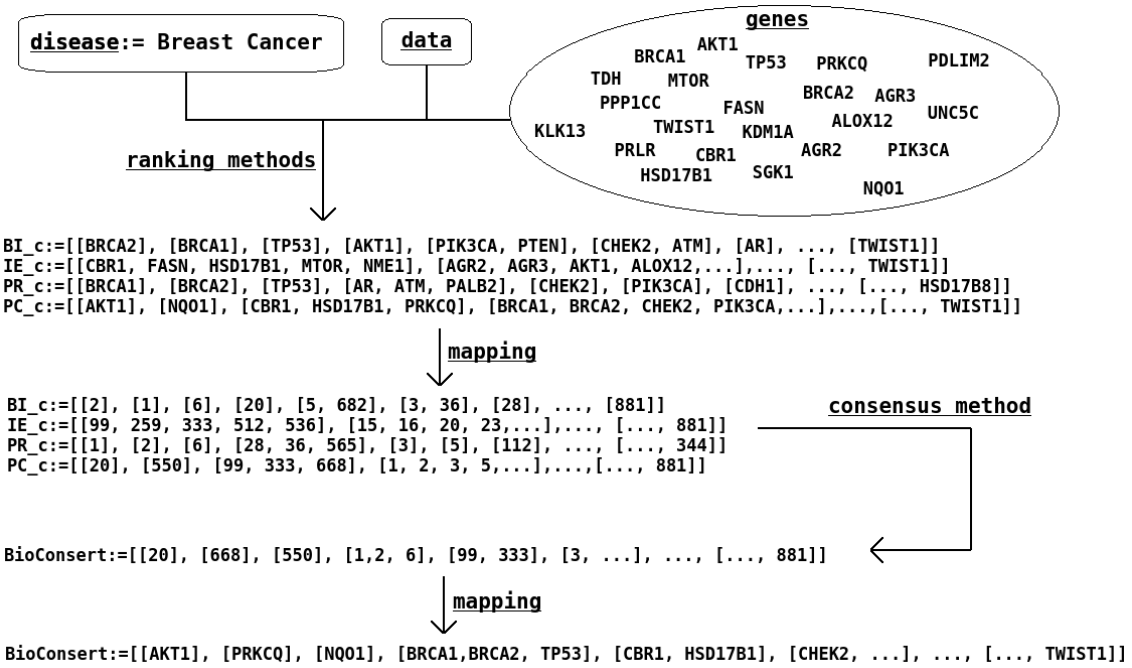


FIGURE 1.5. Exemple de classements de gènes. Dans cette figure, des chercheurs sont intéressés par les gènes associés à la maladie du cancer du sein (*Breast Cancer*). Quatre méthodes de classements de gènes sont utilisées : BI=Bioggle, IE=In Edge, PR=Page Rank et PC=Path Count, ce qui donne quatre classements dans lesquels des ensembles de gènes sont ordonnés selon l'importance prédite par chacune des méthodes. Les chercheurs sont intéressés à obtenir un classement qui décrit le mieux possible ces quatre classements. Le problème est modélisé en associant un entier à chaque gène et le consensus de ces classements est trouvé en utilisant BioConsert. Le classement consensus peut être utilisé pour orienter les chercheurs dans les gènes à investiguer. Cet exemple provient de données provenant du site du projet BioGuide http://bioguide-project.net/bioconsert/.

1.3. Définitions et notations sur les classements

1.3.1. Classements et ensembles de classements

Un classement est une généralisation d'une permutation dans laquelle on introduit la relation d'égalité entre deux éléments. Lorsque des éléments sont à égalité, ils partagent la même position.

Définition 1.3.1. *Soit U l'univers des éléments considérés. Un* **classement** *R est une partition ordonnée d'un sous-ensemble d'éléments $V \subseteq U$. Si $V = U$, le classement est dit* **complet**, *autrement, il est dit* **incomplet**.

Un classement R de n éléments et k positions ($n \leq k$) peut être aussi vu comme une surjection de l'ensemble $\{e_1,e_2,\ldots,e_n\}$ sur l'ensemble $\{1,2,\ldots,k\}$.

On dénote un classement R de $\{e_1,e_2,\ldots,e_n\}$ par $R = \beta_1\beta_2\ldots\beta_n$, où β_j dénote le panier à la position j. Chaque panier représente un ensemble non-vide d'éléments de $\{1,2,\ldots,n\}$. La position $R[i]$ d'un élément i dans le classement R est la position j du panier β_j qui le contient. Un exemple de classement se trouve à l'Exemple 1.3.1.

Définition 1.3.2. *L'***ordre entre deux éléments** *d'un classement est définit comme suit : $i \prec_R j$ signifie que i est préféré à j dans le classement R (mathématiquement $R[i] < R[j]$) alors que $i \succ_R j$ signifie que j est préféré à i. Si deux éléments i et j sont à égalité dans R alors on note $i \equiv_R j$ (mathématiquement $R[i] = R[j]$).*

Dans le cas où il n'y a pas d'ambiguïté, la notation $i \prec_R j$ est simplifiée à $i \prec j$ et la notation $i \equiv_R j$ est simplifiée à $i \equiv j$. Chaque paire d'éléments d'un classement a un ordre défini ($i \prec j$, $j \prec j$ ou $i \equiv j$) et l'ordonnancement est transitif :

— $i \prec j, j \prec k \rightarrow i \prec k$

— $i \equiv j, j \equiv k \rightarrow i \equiv k$

— $i \prec j, j \equiv k \rightarrow i \prec k$

— $i \equiv j, j \prec k \rightarrow i \prec k$.

Exemple 1.3.1. *Le classement $R = [[1,3],[5],[6,8,9],[4],[2]]$ est un classement de taille $n = 9$. À la position 3 se trouve le panier β_3 qui contient les éléments 6, 8 et 9. La position de l'élément 3 est $R[3] = 1$. Dans R, l'élément 3 précède l'élément 6, noté $3 \prec 6$, et l'élément 3 est à égalité avec l'élément 1, noté $3 \equiv 1$. Le classement est incomplet car il manque l'élément 7, l'univers étant $U = \{1,2,...,9\}$.*

Définition 1.3.3. L'ensemble de tous les classements complets de taille *n est noté $Rank_n$.*

$Rank_n$ est de taille :

$$|Rank_n| \approx \frac{n!}{2(\log 2)^{n+1}}$$

La formule d'approximation est due à Wilf [**136**] et la séquence des tailles $|Rank_n|$ se retrouve sur le site OEIS https://oeis.org/A000670 nommée "les nombres de Fubini".

Pour caractériser un ensemble de classements $\mathcal{R}$, on définit trois matrices : $L(\mathcal{R})$ la **matrice gauche**, $R(\mathcal{R})$ la **matrice droite** et $E(\mathcal{R})$ la **matrice d'égalité** qui comptent

le nombre de classements dans lesquelles un élément i est à gauche de j ($i \prec j$), est à droite de j ($j \prec i$) ou est à égalité avec j ($i \equiv j$).

Définition 1.3.4. *$L_{ij}(\mathcal{R})$ (resp. $R_{ij}(\mathcal{R})$ et $E_{ij}(\mathcal{R})$) dénote le nombre de classements de l'ensemble $\mathcal{R}$ où l'élément i est à gauche de l'élément j, $i \prec j$ (resp. i à droite de j, $j \prec i$ et i à égalité avec j, $j \equiv i$), $1 \leq i \neq j \leq n$:*

$$L_{ij}(\mathcal{R}) = \#\{T \in \mathcal{R} \mid T[i] < T[j]\},$$
$$R_{ij}(\mathcal{R}) = \#\{T \in \mathcal{R} \mid T[i] > T[j]\}, \text{ et}$$
$$E_{ij}(\mathcal{R}) = \#\{T \in \mathcal{R} \mid T[i] = T[j]\}.$$

Par définition on a que $L_{ij}(\mathcal{R}) + R_{ij}(\mathcal{R}) + E_{ij}(\mathcal{R}) = m = |\mathcal{R}|$, si tous les classements de $\mathcal{R}$ sont complets.

1.3.2. Distance de Kendall-τ généralisée

Comme avec le cas des permutations, la distance de Kendall-τ généralisée entre deux classements va compter le nombre de paires d'éléments qui sont en désaccord par rapport à leurs ordres ($i \prec j$, $j \prec i$ ou $i \equiv j$) dans les deux classements.

Définition 1.3.5. *La* **distance de Kendall-τ généralisée**, *avec une pénalité p, entre deux classements complets R et T compte le nombre de paires d'éléments qui sont en désaccord par rapport à leur ordre :*

$$\begin{aligned} K^{(p)}(R,T) \;=\; & \#\{(i,j) : i < j \quad \wedge [(R[i] < R[j] \;\wedge\; T[i] > T[j]), \vee \\ & \qquad\qquad (R[i] > R[j] \;\wedge\; T[i] < T[j]))\} \\ & + p \times \#\{(i,j) : i < j \quad \wedge (R[i] = R[j] \;\wedge\; T[i] \neq T[j]), \vee \\ & \qquad\qquad (R[i] \neq R[j] \;\wedge\; T[i] = T[j])]\} \end{aligned}$$

Ici, R et T sont deux classements de même taille n. La relation $i \neq j$ signifie que les éléments i et j ne sont pas à égalité, donc soit $i \prec j$ ou $j \prec i$. La variable $p \in [0.5,1]$ permet de paramétrer la pénalité appliquée à une différence entre un ordre d'égalité et un ordre gauche/droite. Dans notre cas, on va poser $p = 1$ et étudier le cas plus général plus tard. On utilisera $K(R,T)$ à la place de $K^{(p)}(R,T)$ lorsqu'il n'y a pas d'ambiguïté.

Exemple 1.3.2. *Distance de Kendall-τ généralisée entre deux classements R et T avec $p = 1$. Les paires en rouges représentent les désaccords et leur nombre (4) est la distance*

de Kendall-τ généralisée (dans le cas général, la distance est de $2 + 2 \times p$) :

R = [[1,2],[4],[3],[5]]
T = [[1,2,3],[5],[4]]

(1,2)	**(2,3)**	**(3,4)**	**(4,5)**
(1,3)	*(2,4)*	*(3,5)*	
(1,4)	*(2,5)*		
(1,5)			

1.3.3. Score de Kemeny généralisé et problème du consensus de classements

Définition 1.3.6. *Le* **score de Kemeny généralisé** *est définit entre un classement complet et un ensemble de classements complets : Soit un classement $R \in Rank_n$ et un ensemble de classements $\mathcal{R} \subseteq Rank_n$ alors :*

$$K^{(p)}(R, \mathcal{R}) = \sum_{R_i \in \mathcal{R}} K^{(p)}(R, R_i).$$

Définition 1.3.7. *Le* **problème du consensus de classements** *(aussi* **problème de la médiane de classements***) se définit mathématiquement comme suit :*
Soit un ensemble de classements complets $\mathcal{R} = \{R_1, \dots R_t\}$, $R_i \in Rank_n$, et une pénalité $p \in [0.5,1]$, trouver l'ensemble des classements complets $R^* \in Rank_n$ tel que :

$$K^{(p)}(R^*,\mathcal{R}) \leq K^{(p)}(R,\mathcal{R}), \forall R \in Rank_n.$$

Cet ensemble de classements médians est dénoté $M(\mathcal{R})$.

1.3.4. Modélisation en programmation en nombres entiers de la généralisation

Comme avec le problème de la médiane de permutations, on peut aussi modéliser le problème du consensus de classements en programmation en nombres entiers (Integer Linear Programming ou ILP) en modifiant le modèle basé sur les permutations (Voir Section 1.1.3). Le modèle suivant est apparut dans [**31**] :

$$minimiser : \sum_{i<j,\ i,j\in\{1,\dots,n\}} \Big(R_{ij}(\mathcal{R})+E_{ij}(\mathcal{R})\Big)x_{i<j}+\Big(L_{ij}(\mathcal{R})+E_{ij}(\mathcal{R})\Big)x_{i>j}+p\Big(R_{ij}(\mathcal{R})+L_{ij}(\mathcal{R})\Big)x_{i\equiv j} \tag{1.3.1}$$

sujet aux contraintes :

$$x_{i<j} + x_{i>j} + x_{i\equiv j} = 1, \quad i \neq j,\ i,j \in \{1,...,n\} \tag{1.3.2}$$

$$x_{i<k} - x_{i<j} - x_{j<k} \geq -1, \quad i \neq j \neq k,\ i,j,k \in \{1,...,n\} \tag{1.3.3}$$

$$2x_{i<j} + 2x_{i>j} + 2x_{j<k} + 2x_{j>k} - x_{i<k} - x_{i>k} \geq 0, \quad i \neq j \neq k,\ i,j,k \in \{1,...,n\} \tag{1.3.4}$$

$$x_{i<j}, x_{i>j}, x_{i\equiv j} \in \{0,1\} \quad i < j,\ i,j \in \{1,...,n\} \tag{1.3.5}$$

Les variables $x_{i<j}, x_{i>j}, x_{i\equiv j}$ définissent l'ordre entre les éléments i et j : si $x_{i<j} = 1$ alors i précède j dans la solution, si $x_{i>j} = 1$ alors i succède j dans la solution puis si $x_{i\equiv j}$ alors i et j sont dans le même panier dans la solution. Notez un détail de l'implémentation : les variables $x_{i<j}$ et $x_{j>i}$ représentent la même décision alors, lors de l'implémentation, il faut soit ajouter la contrainte $x_{i<j} = x_{j>i}$, soit gérer le modèle pour que seulement une des deux variables soit utilisée dans les contraintes et la fonction objective. Le deuxième choix est plus efficace.

Dans la fonction de coût (1.3.1), $R_{ij}(\mathcal{R})$ indique le nombre de classements dans $\mathcal{R}$ dans lesquelles i est à droite de j, de même pour $L_{ij}(\mathcal{R})$ et $E_{ij}(\mathcal{R})$, alors si i précède j dans la solution ($x_{i<j} = 1$), on doit additionner $R_{ij}(\mathcal{R})$ et $E_{ij}(\mathcal{R})$ qui comptent le nombre de classements en désaccords avec cette paire. Il va de façon similaire pour $x_{i>j} = 1$ et $x_{i\equiv j} = 1$. On applique une pénalité p au paires qui "brisent" les égalités. La contrainte (1.3.2) $x_{i<j} + x_{i>j} + x_{i\equiv j} = 1$ force à avoir un ordre soit $i \prec j$ soit $i \succ j$ soit $i \equiv j$.

Les contraintes (1.3.3) et (1.3.4) forcent la transitivité entre les ordres des éléments. La contrainte (1.3.3) $x_{i<k} - x_{i<j} - x_{j<k} \geq -1$ force la transitivité avec les ordres avant et après ($\prec$ et $\succ$). Si $i \prec j$ et $j \prec k$ alors on doit avoir $i \prec k$. La contrainte (1.3.4) $2x_{i<j} + 2x_{i>j} + 2x_{j<k} + 2x_{j>k} - x_{i<k} - x_{i>k} \geq 0$ force la transitivité avec les ordres d'égalité ($i \equiv j$). Si $i \equiv j$ et $j \equiv k$ alors on doit avoir $i \equiv k$ pour que tous ces éléments soient dans le même panier. Finalement, la contrainte (1.3.5) $x_{i<j}, x_{i>j}, x_{i\equiv j} \in \{0,1\}$ oblige les variables d'ordre à être binaires, alors on a soit $i \prec j$ ou $i \succ j$ ou $i \equiv j$.

Encore une fois comme dans le cas des permutations, la résolution de ce problème ILP donne le score optimal de Kemeny et des assignations de variables que l'on peut convertir en un classement consensus pour l'ensemble $\mathcal{R}$. Différents logiciels commerciaux ou open source sont disponibles pour résoudre cette modélisation (Voir Section 1.1.3). Si la dernière contrainte (contrainte d'intégralité) est relaxée $x_{ij} \in [0,1]$, une borne inférieure est obtenue par la méthode du simplexe. Il est à noter que seules les matrices $L(\mathcal{R})$, $R(\mathcal{R})$ et $E(\mathcal{R})$ suffisent pour cette modélisation.

1.3.5. Autres généralisations

Le problème du consensus de classements avec égalité est une généralisation de la médiane de permutations. Si le cas où les classements sont complets et la pénalité de "casser" une égalité est $p = 1$ est la première généralisation qui vient intuitivement, il existe plusieurs autres généralisations.

Une première généralisation possible est d'attribuer une pénalité p différente de 1 dans le calcul de la distance de Kendall-τ généralisée pour un désaccord entre un ordre d'égalité et un ordre avant/après. Cela vient de l'idée que ce désaccord est moins fort que celui entre un ordre "avant" et un ordre "après". Dans [21], une pénalité de $p = \frac{1}{2}$ est utilisée.

Une seconde généralisation est de considérer les classements incomplets. Ce cas est très pratique dans les applications et plusieurs auteurs l'utilisent. On peut s'imaginer plusieurs courses dans lesquelles participent des athlètes mais ces athlètes ne participent pas obligatoirement à toutes les courses. Un autre exemple serait le cas des films où des utilisateurs qui n'ont vu qu'un sous-ensemble d'une grande collection jugent avec un système de classements basé sur 5 étoiles. Présentement, les façons de gérer les candidats absents sont de normaliser les classements. La première méthode de normalisation est l'**unification**, qui consiste à ajouter les candidats absents à la fin de chaque classement dans un panier. La seconde méthode de normalisation est la **projection**, qui consiste à ne considérer que les candidats présents dans tous les classements et d'ignorer les autres. Ces méthodes sont utilisées dans [18, 31] entre autres.

On peut noter cependant que les deux méthodes peuvent s'avérer injustes pour un contexte sportif, par exemple, car un athlète pourrait être fortement pénalisé pour une absence ou simplement éliminé. Deux autres généralisations qui visent à amoindrir cette dernière observation se retrouvent dans la littérature. Dans [30], le panier rajouté par le procédé d'unification, nommé **panier d'unification**, est traité différement des autres paniers. Alors que dans [56], on ne minimise que les désaccords avec les paires existantes dans les classements. Cette généralisation de la distance de Kendall-τ est nommée **distance de Kendall-τ induite** dans [28] (*induced Kendall-τ distance* dans [56]) alors que la fonction de score de Kemeny est nommée **score de Kemeny induit**. Dans [21], la distance de Kendall-τ induite est utilisée avec $p = \frac{1}{2}$.

Une autre généralisation est celle retrouvée dans [2] où les égalités sont considérées comme un manque d'information. La distance de Kendall-τ généralisée compte seulement les paires d'éléments dont l'ordre est inversé ($i \prec j$ vs $j \prec i$) et ignore les paires d'éléments à égalité.

Récemment (2017), Pierre Andrieu, un étudiant au doctorat au Laboratoire de Recherche en Informatique à l'Université Paris-Sud, a fondé une forte base théorique qui permet d'englober toutes les généralisations connues de la littérature. Dans ce cadre, il définit six situations possibles pour une paire d'éléments de l'univers U dans un classement R qui peut être incomplet : i précède j ($i \prec j$), i succède j ($i \succ j$), i est a égalité avec j ($i \equiv j$), i est présent mais pas j ($i!j$), j est présent mais pas i ($!ij$) et les deux éléments i et j sont absents du classement ($!i!j$). Des vecteurs associent une pénalité (ou score) pour chaque situation possible pour une paire d'éléments entre deux classements. Le score de Kemeny généralisé somme ces pénalités sur toutes les paires. Les fonctions de score qui peuvent être représentées par ce modèle sont dites *Kemeny compatibles*. Cette voie est investiguée davantage dans le Chapitre 7.

On peut définir maintenant le problème général du consensus de classements dans lequel les classements peuvent être incomplets et la fonction de score de Kemeny peut être paramétrée arbitrairement.

Définition 1.3.8. *Le* **problème général du consensus de classements** *se définit mathématiquement comme suit :*
Soit un ensemble de classements $\mathcal{R} = \{R_1, \dots R_t\}$, $R_i \in Rank_n$ *, et une fonction* K *de score entre 2 classements, trouver l'ensemble des classements complets* $R^* \in Rank_n$ *tel que :*

$$K(R^*,\mathcal{R}) \leq K(R,\mathcal{R}), \forall R \in Rank_n.$$

Plusieurs généralisations du problème de la médiane de permutations ont été présentées dans cette section. Le choix de la généralisation à utiliser pour une application donnée requiert une analyse approfondie du contexte de l'application et sort du cadre de cette thèse. Par contre, si ce choix doit être fait, le lecteur est amené à la Section 2.10 et au Chapitre 3 pour s'inspirer de choix qui ont été fait dans quelques applications. Le prochain chapitre présente les travaux qui ont été fait sur le problème de la médiane de permutations et sur le consensus de classements.

Chapitre 2

REVUE DE LITTÉRATURE

La distance de Kendall-τ a été introduite pour la première fois en 1938 par Maurice Kendall [78], un statisticien britannique.

Le problème de la médiane d'un ensemble de m permutations $\mathcal{A} \subseteq S_n$ sous la distance de Kendall-τ, a été introduit par John George Kemeny [76] en 1959, un mathématicien et informaticien avec l'objectif de créer une méthode pour trouver un consensus entre divers ordres de préférence.

Le problème existe sous différents noms dans la littérature :

— problème de la médiane de permutations (*median of permutations problem*)[26, 96],
— méthode de Kemeny-Young (*Kemeny-Young method*)[24, 109, 112, 126, 133],
— règle de Kemeny (*Kemeny rule*) [11, 15, 51, 140, 114],
— règle de classement de Kemeny (*Kemeny ranking rule*)[6],
— problème de Kemeny (*Kemeny's problem*)[36]
— problème de classement de Kemeny (*Kemeny ranking problem*)[6, 12, 72]
— problème d'agrégation de classements de Kemeny (*Kemeny Rank Aggregation problem*) [18, 48, 80, 104],
— agrégation optimale de Kemeny (*Kemeny Optimal Aggregation*) [1, 120],
— agrégation de Kemeny (*Kemeny Aggregation*) ou p-KAGG [63, 123, 129],
— problème d'agrégation de classements (*Rank Aggregation problem*)[3, 56, 92, 117],
— problème de minimisation de croisements de k permutations (*Crossing Minimization of k Permutations problem*) ou PCM-k [23],
— problème du classement consensus (*Consensus Ranking problem*) [58],
— schéma de vote de Kemeny (*Kemeny's voting scheme*) [72],
— méthode de Condorcet-Kemeny-Young-Levenglick (*Condorcet-Kemeny-Young-Levenglick method*) ou simplement C-K-Y-L [102],
— problème de l'ordre médian (*median order problem*)[70],
et finalement,

— classement par popularité VoteFair (*VoteFair popularity ranking*)
http://www.votefair.org/.

De plus, la médiane porte aussi les noms de **consensus**, **consensus de Kemeny**, **classement optimal de Kemeny** et **agrégation optimale de Kemeny**.

Un graphe de tournoi (Définition 1.1.7) peut être associé à chaque ensemble de permutations. Le problème de la médiane de permutations est alors un cas particulier du problème de *weighted tournament LOP* [**36**] qui lui est un cas particulier du problème général d'ordonnancement linéaire (LOP pour *Linear Ordering Problem*). Ce dernier consiste à trouver un ordonnancement linéaire qui minimise un score calculé à partir d'une matrice de coût. Cette matrice associe une pénalité pour chaque choix d'ordre relatif pour une paire d'éléments. Dans notre cas, la matrice de coût est la matrice $R(\mathcal{A})$ (Définition 1.1.6).

Le problème de la médiane de permutations est aussi un cas particulier du *weighted feedback arc set problem* [**3**, **41**, **51**, **107**] (wFAS) qui consiste à trouver un ensemble d'arcs dont la somme est de poids minimale pour rendre un graphe orienté acyclique. Le *weighted tournament feedback arc set problem* et le *weighted tournament LOP* sont des synonymes.

Donc, tous les travaux qui ont trait au *weighted tournament LOP*, au *LOP* et au *weighted feedback arc set problem* sont aussi applicables dans notre cas. Nous nous limiterons par contre, dans cette revue de littérature, aux travaux qui sont directement concentrés sur le problème de la médiane pour des raisons de temps et d'espace. À l'occasion quelques travaux très pertinents de LOP ou de wFAS seront mentionnés.

Dans ce qui suit, nous ferons un résumé des travaux antérieurs réalisés dans plusieurs directions de recherche. Ces travaux on trait principalement à l'approximation d'une médiane, aux algorithmes à paramètres fixes, à la résolution exacte du problème, à l'étude de la complexité du problème, des bornes inférieures et de la réduction d'espace de recherche. Nous discuterons brièvement des données utilisées dans ces travaux à la fin du chapitre.

2.1. Système électoral

En science politique, le problème de la médiane de permutations est vu comme un système électoral où m électeurs vont individuellement ordonner une liste de n candidats selon leurs préférences. La méthode de Kemeny-Young est parfois catégorisée comme fonction de choix social.

Dans ce contexte, des chercheurs se sont intéressés aux propriétés mathématiques de ce système électoral, aussi nommées "critères". Ces critères sont la base pour comparer différents systèmes électoraux et pour justifier le choix d'un système électoral dans une certaine application. Ils sont aussi pratiques pour avoir une bonne compréhension d'un certain système électoral.

2.1.1. Propriétés mathématiques de systèmes électoraux

La liste suivante présente une sélection pertinente de critères de systèmes électoraux en ordre alphabétique avec une courte description pour chacun. Leur applicabilité dans la méthode de Kemeny-Young, notée entre parenthèse (**Oui**/**Non**) est brièvement discutée.

— 1) Critère d'**anonymité** [**140**] : (**Oui**) Le résultat de l'élection dépends seulement des votes et pas de leur provenance (électeurs). Ce critère est respecté par la description même de la méthode de Kemeny-Young où aucune permutation n'est traitée différement des autres.

— 2) Critère de **calculabilité** : (**Oui**) Le résultat de l'élection a peu de chance d'avoir des égalités. Dans [**130**], ce critère est définit comme le fait que chaque égalité peut être rompue par un seul vote additionnel.

— 3) Critère de **Condorcet** [**43**] : (**Oui**) Si un candidat est préféré à tous les autres candidats dans une comparaison un à un, alors ce candidat est le vainqueur de l'élection. En fait, la méthode de Kemeny-Young fait partie de la famille des méthodes de Condorcet, qui regroupe toutes les méthodes respectant le critère de Condorcet.

— 4) Critère de **consistance** ou de **séparabilité** [**125**] : (**Non**) Si les votes d'une élection sont séparés en plusieurs groupes et que le même candidat est vainqueur dans chacun de ces groupes alors il est vainqueur dans l'élection. Dans [**137**], il a été montré que seulement les systèmes électoraux préférentiels à base de score sont séparables et donc la méthode de Kemeny-Young n'est pas séparable. Notez qu'on considère que le candidat vainqueur à la différence du critère 14 qui considère l'ordre de tous les candidats.

— 5) Critère de **domaine non-restreint** ou **universalité** : (**Oui**) Toutes les options de vote possibles peuvent être choisies par un électeur. La méthode de Kemeny-Young respecte ce critère car il n'y a aucune restriction sur les permutations de l'ensemble $\mathcal{A}$.

— 6) Critère d'**indépendance des alternatives non-pertinentes** : (**Non**) La préférence entre deux candidats n'est pas affectée par un troisième candidat. Par le théorème d'impossibilité d'Arrow [**9**] (voir Section 2.1.2.1), il peut être démontré que Kemeny-Young ne respecte pas ce critère. Intuitivement, la méthode de Kemeny-Young qui tente de minimiser les désaccords, considère simultanément toutes les relations entre les paires de candidats pour trouver le consensus, donc la dépendance est clairement présente.

— 7) Critère d'**indépendance locale des alternatives non-pertinentes** [**139**] : (**Oui**) Si le vainqueur ou le dernier perdant est retiré d'une élection alors le reste du résultat ne doit pas changer. Dans une optique d'optimisation, on peut imaginer par contradiction, que le premier élément (ou le dernier) est enlevé et que le reste du résultat change. Donc, ce segment de la permutation "médiane" n'était pas optimal et donc le résultat n'était pas une permutation médiane. Ce critère est une conséquence de la propriété de stabilité locale discutée à la Section 2.1.2.2 et dans [**138**]. Notez que ce critère est confondu avec le critère précédant dans [**103**].

— 8) Critère de **majorité** : (**Oui**) Si la majorité des électeurs préfèrent un candidat à tous les autres candidats alors ce candidat doit être le vainqueur. Toute méthode de Condorcet respecte le critère de majorité, dont la méthode de Kemeny-Young.

— 9) Critère de **neutralité** [**140**] : (**Oui**) Le résultat de l'élection ne dépends pas des noms des candidats ou de l'ordre dans lequel ils sont présentés, i.e. tous les candidats sont traités de la même façon. Ce critère est respecté par la description même de la méthode de Kemeny-Young où aucun candidat n'est traité différement des autres.

— 10) Critère de **non-dictature** : (**Oui**) Aucun électeur n'a le pouvoir total sur l'élection. Ce critère est respecté par la description même de la méthode de Kemeny-Young où aucun vote n'est traité différement des autres.

— 11) Critère de **non-imposition** : (**Oui**) Tous les résultats d'élections sont possibles. Pour la méthode de Kemeny-Young, il suffit qu'une majorité d'électeurs votent tous π pour que le résultat de l'élection soit π, pour n'importe quelle permutation $\pi \in S_n$, grâce au critère de domaine non-restreint.

— 12) Critère de **Pareto** ou **unanimité** [**138**] : (**Oui**) Si tous les électeurs préfèrent un premier candidat à un second alors cette préférence se retrouve dans le résultat de l'élection. Une preuve de ce critère est donnée par le *Always Theorem* [**26**] présenté à la Section 2.7.

— 13) Critère de **participation** : (**Non**) Un ou plusieurs électeurs ne peuvent pas favoriser un candidat en s'abstenant de voter leurs préférences. Autrement dit, ajouter à l'élection un vote qui préfère un premier candidat à un second ne peut changer le résultat de l'élection tel que le premier candidat perd sa victoire en faveur du second. La méthode de Kemeny-Young ne respecte pas ce critère. Un contre-exemple peut être trouvé dans [**62**].

— 14) Critère de **renforcement** [**139**] : (**Oui**) Si les votes d'une élection sont séparés en plusieurs groupes et que le résultat électoral est le même pour chaque groupe alors il doit être le même pour l'élection. Young et Levenglick montrent dans [**140**] que Kemeny-Young respecte ce critère (nommé **consistance**, mais ensuite **renforcement** dans [**139**]). Intuitivement, si une même permutation minimise le score de Kemeny pour chacun de ces groupes, le choix d'une autre permutation ne pourrait qu'augmenter le score de Kemeny pour chacun de ces groupes et pour l'élection.

— 15) Critère de **symétrie inverse** : (**Oui**) Si un candidat est le vainqueur d'une élection et que tous les votes sont inversés, alors ce candidat ne doit pas être vainqueur dans l'élection inversée. Soit $\mathcal{A} \subseteq S_n$ un ensemble de permutations. Il suffit de voir dans un optique d'optimisation que pour minimiser le score de Kemeny de l'ensemble $\mathcal{A}^r = \{\pi^r \mid \pi \in \mathcal{A}\}$, où $\pi^r = \pi_n\pi_{n-1}...\pi_2\pi_1$, il suffit de renverser la permutation qui minimise le score pour $\mathcal{A}$: $K(\pi^*,\mathcal{A}^r) = K(\pi^{*r},\mathcal{A})$. La preuve est présentée par le Corollaire A.0.1 de l'annexe A.

— 16) Critère de **temps de calcul polynomial** : (**Non**) Le résultat de l'élection peut être calculé en temps polynomial. La méthode Kemeny-Young ne satisfait pas ce critère [**14**], ce qui n'est pas très souhaitable. Par contre, il est intéressant de souligner que le résultat est quand même calculable par opposition à certains systèmes électoraux probabilistes. De plus, pour un petit nombre de candidats ($n \leq 30$), une médiane peut être facilement calculée [**97**].

Les premiers travaux sur le problème de la médiane de permutations ont été réalisés dans le contexte du système électoral et sont présentés dans la sous-section qui suit.

2.1.2. Travaux liés aux systèmes électoraux

2.1.2.1. *Travaux de Kenneth Arrow*

Le livre "Social Choice and Individual Values" [**9**] de Kenneth Arrow publié en 1970, qui ne traite pas directement de Kemeny-Young mais qui y est fortement lié, a été un pilier de la théorie du choix social. Dans cet ouvrage, l'auteur introduit un théorème,

mieux connu sous le nom du "théorème d'impossibilité d'Arrow" dans lequel il explique qu'il est impossible d'avoir un système électoral avec ordonnancement des candidats (et avec domaine non-restreint) qui respecte les trois critères suivants : A) non-dictature, B) Pareto, et C) indépendance des alternatives non-pertinentes. Dans le cas de la méthode de Kemeny-Young, c'est le critère C) qui n'est pas satisfait.

2.1.2.2. *Travaux de Peyton Young*

Hobart Peyton Young a réalisé dans [**140**] les premiers travaux sur le problème de la méthode de Kemeny-Young en 1978 avec Arthur Levenglick ; d'où le nom de la méthode. Ils montrent que ce système électoral est le seul qui respecte le critère de renforcement (critère #14) (nommé **consistance** dans leur travail), le critère de neutralité (critère #9) et le critère de Condorcet [**43**] (critère #3). Ils montrent aussi que le système est anonyme (critère #1).

Young poursuit dans un travail très important, en montrant qu'une permutation médiane est un estimateur de maximum de vraisemblance selon un modèle où les permutations de l'ensemble de départ sont des permutations obtenues par perturbation à partir d'une "vraie" permutation [**138**]. Une interprétation intuitive est celle où l'on voit les permutations de l'ensemble $\mathcal{A}$ comme des observations bruitées d'une réalité sous-jacente et le consensus de Kemeny permet d'estimer cette réalité. Dans ce modèle, introduit par Condorcet en 1785 [**43**], les électeurs choisissent, pour chaque paire de candidats, le meilleur candidat avec une même probabilité $\frac{1}{2} < p < 1$. Ce modèle est montré équivalent [**6**] au modèle **Mallows-ϕ** de Mallows [**89**].

Young montre aussi, dans ce travail, que la méthode de Kemeny-Young est **localement stable** : Si on considère un segment dans une permutation médiane, alors la projection de l'ensemble des candidats de ce segment dans une élection doit donner un résultat identique au segment. Dans le cas où plusieurs médianes sont possibles pour le problème projeté, le segment peut être changé par chacune des médianes du problème projeté et la permutation contenant ce segment reste une médiane pour l'élection générale.

Dans [**139**], Young défend la médiane comme meilleur consensus contre la moyenne (qui serait la permutation qui minimise la somme des distances au carré). Il désigne parfois la méthode de Kemeny-Young comme la "méthode de maximum de vraisemblance" en référence au modèle de Mallows. De plus, il montre que la méthode de Kemeny-Young est localement indépendante des alternatives non-pertinentes (critère #7).

Cette série de travaux est très importante car elle offre une base théorique sur laquelle on peut s'appuyer pour justifier le choix de la méthode de Kemeny-Young dans une application. En effet, plusieurs auteurs d'articles d'application du problème (voir Section 3) justifient l'utilisation de Kemeny-Young avec cette base. Parallèlement, plusieurs autres auteurs justifient son utilisation avec une approche empirique.

Après ces travaux pionniers sur le problème, plusieurs années ont passées et le problème est revenu en force à l'attention de la communauté informatique depuis les 20 dernières années.

2.2. Complexité du problème

La complexité du problème de la médiane de permutations a été investiguée pour la première fois dans [14]. Il est montré que le problème de décision *Kemeny Score* (Problème de décision 1.1.1) est NP-complet et que *Kemeny Winner* (Problème de décision 1.1.2) et *Kemeny Ranking* (Problème de décision 1.1.3) sont NP-difficile. *Kemeny Score* est démontré NP-complet grâce à une réduction à partir du problème de décision du *Feedback Arc Set* [74] (FAS), qui est NP-complet, vers *Kemeny Score*. Le problème de décision du FAS demande s'il existe un ensemble d'arcs E à retirer d'un graphe orienté G pour le rendre acyclique tel que $|E| \leq k$ pour un k donné. Pour la réduction, il suffit de créer un certain ensemble V d'électeurs qui votent tous deux fois. Chaque électeur aura un arc du FAS assigné uniquement à lui et les sommets du FAS seront des candidats d'élection. Les deux votes de chaque électeur seront inversés un par rapport à l'autre sauf pour la paire de sommets de l'arc (i,j) assigné qui partageront la même préférence $i \prec j$ dans les deux votes. Une permutation représente un ordre stricte (sans cycle dans le graphe) et les paires d'éléments de la permutation qui ne sont pas en accord dans leur ordre avec le graphe orienté G (a précède b dans la permutation mais l'arc dans le graphe est orienté (b,a)), correspondent aux arc à retirer dans le FAS. Donc s'il existe une permutation avec un score de Kemeny de $|V|(|V|-1)((|V|-2)/2)+4k$, alors il existe un ensemble E d'arcs, de taille $|E| = k$, à retirer pour satisfaire le FAS. Notez que le problème du Minimum Feedback Arc Set, qui consiste à trouver l'ensemble E de taille minimale, est NP-difficile contrairement au problème de décision du FAS avec une taille donnée de l'ensemble E. Les auteurs disent qu'il suffit de faire quelques modifications simples de la preuve pour montrer que *Kemeny Winner* et *Kemeny Ranking* sont NP-difficiles.

Une version du problème Kemeny Score où les permutations sont pondérées par des nombres rationnels est démontrée NP-complète dans [39] pour un nombre d'élément $n \geq 3$. Cette pondération revient à avoir des multiensembles de permutations.

Dans [56, 57], Dwork et al. montrent, pour un ensemble de m permutations, $m \geq 4$ et m pair, que le problème *Kemeny Ranking* est NP-difficile et que *Kemeny Score* est NP-complet. La preuve se base sur la réduction du FAS au problème de la médiane de permutations. Dans la preuve, quatre permutations sont construites à partir du graphe du problème FAS modifié (chaque arc est dupliqué en 2 arcs). Une solution au problème de la médiane sur ces quatre permutations peut être convertie en solution pour le FAS et, inversement. Une petite erreur dans cette dernière preuve a été par la suite corrigée dans [23].

Dans [72], Hemaspaandra et al. investiguent davantage la complexité générale du problème et montrent que les problèmes de décision *Kemeny Winner* et *Kemeny Ranking* sont $P_{\|}^{NP}$-complet. La classe $P_{\|}^{NP}$ [84] est la classe de complexité où la machine de Turing peut accéder à un oracle NP de façon parallèle (par opposition à accéder de façon séquentielle, les réponses de l'oracle ne peuvent pas influencer les prochaines réponses). On a aussi que $P_{\|}^{NP} \subseteq P^{NP}$. Un corollaire de leur théorème dit que si *Kemeny Winner* est NP-complet alors $PH = NP$, qui est une question ouverte. PH est l'union des classes de la hiérarchie polynomiale, donc inclus P, NP, $coNP$, P^{NP} et les classes supérieures. Une conséquence de $PH = NP$ serait que $NP = coNP$. Il est donc possible mais peu probable que le problème *Kemeny Winner* soit NP-complet.

Alors que la complexité pour les cas où l'ensemble de permutations est de cardinalité impaire a été longtemps inconnue, il été suggéré récemment que le problème *Kemeny Ranking* est NP-difficile pour $m \geq 7$, m impair dans [11].

La complexité de *Kemeny Ranking* pour les cas $m = 3$ et $m = 5$ demeure inconnue.

2.3. Approches exactes pour résoudre le problème

Alors qu'une multitude de méthodes heuristiques pour le problème de la médiane de permutations ont été décrites dans la littérature (voir Section 2.4), plusieurs travaux ont été réalisés pour résoudre le problème exactement. Calculer une solution exacte au problème n'est pas seulement d'intérêt mathématique ou informatique. Dans le cadre d'élections, il est évidemment important de trouver une médiane (et peut-être toutes les médianes) étant donné que l'utilisation de n'importe quelle autre heuristique reviendrait à changer le système électoral. De plus, avoir une solution optimale donne la garantie que le

consensus est bien l'estimateur du maximum de vraisemblance de la permutation centrale de $\mathcal{A}$ selon le modèle de Young (voir Section 2.1).

Les deux principales approches exactes qui ont été étudiées sont l'approche branch-and-bound et l'approche par programmation linéaire en nombres entiers. Une troisième approche est la recherche de classes spéciales du problème qui sont faciles à résoudre.

2.3.1. Branch-and-bound (BnB)

Un premier algorithme branch-and-bound pour le problème est introduit dans [**51**]. Dans cet algorithme, une recherche en profondeur est effectuée et chaque noeud de recherche est un ensemble de paires d'éléments qui ont été ordonnés. À l'étape du branchement, une paire d'éléments, dont l'ordre n'a pas encore été déterminé, est choisie puis ordonnée. Pour $\mathcal{A} \subseteq S_n$, les paires (i,j), $1 \leq i \neq j \leq n$, dont la différence entre $R_{ij}(\mathcal{A})$ et $R_{ji}(\mathcal{A})$ (Définition 1.1.6) est la plus élevée sont choisies en priorité. Quand la paire est ordonnée, une étape de fermeture transitive est appliquée. À l'étape d'élagage, une borne inférieure ($LowerBound_0$ de la Section 2.8) est évaluée sur le nouvel ensemble de paires d'éléments ordonnées, et si elle dépasse le score de la meilleure permutation trouvée jusqu'à présent, cette branche de l'algorithme est élaguée. Une permutation est obtenue quand toutes les paires d'éléments sont ordonnées. Des tests sont effectués sur des ensembles de 5, 15, 25 et 35 permutations de 15 éléments, générées aléatoirement de sorte que les permutations de ces ensembles partagent une certaines similarité allant de 50% à 100%. Ils concluent que plus les permutations de l'ensemble de départ sont éloignées, plus il est difficile de calculer une médiane. Pour améliorer leur temps de calcul, les auteurs proposent une heuristique gloutonne à partir de leur algorithme exacte en ne permettant pas le retour en arrière.

Dans [**92**] un second algorithme branch-and-bound est proposé dans lequel les noeuds de l'arbre de recherche sont des préfixes de permutations. L'algorithme énumère alors toutes les permutations en ordre lexicographique. L'opération de branchement est simplement de choisir le prochain élément à insérer dans la permutation en construction. L'opération d'élagage est la borne inférieure améliorée de [**44**]. Le prochain noeud a être sélectionné est celui qui possède la borne inférieure la plus petite (méthode meilleur d'abord). Des tests sont effectués sur des instances aléatoires de taille allant jusqu'à $n = 10$.

Dans [**48**], est présenté le coefficient de corrélation de Kendall entre deux permutations : $\tau(\pi,\sigma) = 1 - \frac{2d_{K\tau}(\pi,\sigma)}{n(n-1)}$. Il y est montré que minimiser le score de Kemeny revient à maximiser la somme des coefficients de corrélations de Kendall. Les auteurs décrivent

un algorithme branch-and-bound (provenant de [58]) qui maximise cette corrélation et dans lequel un noeud est une permutation partielle. Le placement d'un nouvel élément dans n'importe quelle position de cette permutation partielle donne un nouveau branchement. L'exploration de l'arbre est guidée par une permutation donnée en entrée qui va déterminer l'ordre dans lequel les éléments seront considérés. Leur algorithme utilise les heuristiques QUICK et FAST [8] (voir aussi Section 2.4.9) pour trouver ces permutations de guidage. Leur BnB peut résoudre des problèmes de taille ≤ 35 éléments dépendamment de la similarité des permutations de départ. Des tests sont fait sur des données synthétiques et réelles, de taille $5 \leq n \leq 10$.

2.3.2. Programmation en nombres entiers

La formulation en programmation en nombres entiers (IP ou ILP) pour le problème de la médiane (Section 1.1.3) a été introduite dans [44] où ILOG CPLEX est utilisé pour résoudre la formulation IP. Des tests sont effectués sur des instances allant jusqu'à 40 éléments, mais toujours avec des ensembles de 5 permutations.

Dans [18], Betzler et al. utilisent les solveurs commerciaux CPLEX, Gurobi et le solveur open source GLPK pour résoudre le modèle de programmation en nombres entiers pour le problème de la médiane. Ils notent une différence significative entre les solveurs commerciaux et open source en faveurs des premiers. Avec les réductions d'espace de recherche (Section 2.7) ECC [43] et 3/4-Majority Rule [18], ils arrivent à résoudre très rapidement des instances allant jusqu'à $n = 100$ candidats de provenance réelle.

Charon et al. [36] proposent un algorithme branch-and-bound ILP qui utilise la relaxation lagrangienne et une méthode de perturbation (*noising*) qui permet de résoudre le problème LOP sur des graphes de tournois pondérés allant jusqu'à $n = 100$ sommets. Le problème de Kemeny en est un sous-problème. La relaxation lagrangienne consiste à relaxer des contraintes d'un modèle ILP et de les remplacer par une pénalité intégrée dans la fonction objective. La fonction objective se trouve ainsi pénalisée si ces contraintes sont non-respectées. La méthode de perturbation consiste à modifier légèrement la fonction objective et de la résoudre avec des heuristiques, et de répéter cette modification pour faire converger la fonction objective perturbée vers la vraie fonction objective. L'avantage est que les chances qu'une solution reste prise dans un minimum local sont diminuées. Les auteurs remarquent que dans les données provenant de problèmes réels, le niveau de transitivité est proche de 100% (i.e. il y a peu de cycles de longueur 3) et cela facilite grandement la résolution.

2.3.3. Classes spéciales du problème

Outre les approches algorithmiques, une approche basée sur l'identification des classes d'instances faciles à résoudre a été étudiée. Elle prend en compte que même si un problème est NP-Difficile, ce ne sont pas toutes les instances qui sont difficiles à résoudre.

Dans [71], travail réalisé pendant ma maîtrise, nous avons investigué deux classes d'instances du problème dites automédianes i.e. les cas où l'ensemble $\mathcal{A}$ est égal à son ensemble de médiane $M(\mathcal{A})$ ($\mathcal{A} = M(\mathcal{A})$). Les instances de la première classe sont caractérisées par le fait que l'ensemble des permutations $\mathcal{A}$ est composé d'une permutation et de toutes les permutations obtenues de celle-ci par "rotation" des éléments. Pour $\pi = \pi_1\pi_2...\pi_n$, l'ensemble de cette classe à partir de π est $\mathcal{A} = \{\pi, \pi_2\pi_3...\pi_n\pi_1, \pi_3\pi_4...\pi_n\pi_1\pi_2,...,\pi_n\pi_1...\pi_{n-1}\}$. La seconde classe comprend les instances dans lesquelles $\mathcal{A}$ est l'ensemble S_n ou contient un noyau S_k avec des éléments points fixes. Des opérations de combinaison de ces classes sont présentées et conservent la propriété d'automédiane.

Dans [53], Charles Desharnais et Sylvie Hamel poussent plus loin l'étude sur les classes automédianes. Premièrement, il est montré que pour $\mathcal{A} \subseteq S_n$ et $\pi \in S_n$, $\mathcal{A} = M(\mathcal{A})$ implique que $\pi M(\mathcal{A}) = M(\pi\mathcal{A})$ où $\pi\mathcal{A}$ représente la composition à gauche de π avec chaque élément de $\mathcal{A}$. Il est aussi montré que la distance de Kendall-τ est invariante sous l'action de composition à gauche i.e. que $\forall\pi,\sigma,\gamma \in S_n$, $d_{KT}(\gamma\pi, \gamma\sigma) = d_{KT}(\pi, \sigma)$. Intuitivement, cette action a pour effet de renommer les éléments des permutations. Deuxièmement, il est montré que la classe des ensembles automédians est fermée sous l'opération de somme directe $\oplus$, i.e. A et B sont des ensembles automédians si et seulement si $A \oplus B$ est automédian. À noter que $A \oplus B = \{\pi \oplus \sigma | \pi \in A, \sigma \in B\}$ où $\pi \oplus \sigma = \pi_1\pi_2...\pi_k\sigma_1 + k...\sigma_l + k$. Finalement, une formule est donnée pour compter le nombre d'ensembles automédians construit par les méthodes précédentes pour une certaine taille n de permutation.

Pour conclure cette section, notez que l'approche branch-and-bound a été étudiée par plusieurs auteurs alors que l'approche par programmation en nombres entiers semble sous-représentée et pourrait bénéficier de plus d'attention étant donné ses bonnes performances en temps d'exécution et en taille d'instance observées dans la pratique. Ceci dit, la prochaine section va aborder les heuristiques.

2.4. Méthodes heuristiques

Si dans le cas du système électoral, il est important de calculer une médiane exacte d'un ensemble de permutations, dans de nombreuses applications il n'est pas nécessaire

d'avoir l'exactitude et une approximation est amplement suffisante. On utilise alors des heuristiques et des métaheuristiques.

Les heuristiques sont des algorithmes qui sont conçus pour trouver rapidement des solutions à un problème d'optimisation spécifique aux dépends de la garantie d'optimalité de la solution. Les métaheuristiques sont des méthodes plus générales applicables à plusieurs problèmes d'optimisation.

Cette section va détailler les heuristiques et métaheuristiques qui ont été proposées et testées pour le problème de la médiane de permutations. Une liste de ces heuristiques peut être trouvée à la Table 2. III de la Section 2.9 qui traite de travaux de comparaison. Alors qu'Ali et Meila [6] ont séparé les heuristiques selon les catégories "à base de tri, à base de graphe, autres heuristiques, combinaisons d'heuristiques et recherche locale", et que Schalekamp et von Zuylen [117] ont séparé les heuristiques selon les catégories " à base de position, à base de comparaison, recherche locale, heuristiques hybrides et combinaisons d'heuristiques", les heuristiques sont classées ici dans 11 catégories correspondant chacune à une section. Les combinaisons d'heuristiques ne sont pas traitées ici, mais la dernière catégorie relève des heuristiques de raffinement qui peuvent être appliquées au résultat de n'importe quelle autre heuristique.

2.4.1. Heuristiques naïves

Les auteurs de [3, 117] ont démontré que 2 heuristiques très simples sont aussi des 2-approximations, c'est-à-dire que le score de Kemeny de la solution trouvée à l'ensemble de départ est au plus 2 fois plus grand que le score de Kemeny d'une vraie médiane. La première heuristique **Pick-a-perm** [7, 3] consiste à choisir comme médiane (approximée) d'un ensemble de permutations $\mathcal{A} \subseteq S_n$, n'importe laquelle des permutations choisie aléatoirement dans $\mathcal{A}$, alors que sa version déterministe, **BestOfA** [117], consiste à choisir comme médiane (approximée) la permutation de $\mathcal{A}$ qui minimise la distance à $\mathcal{A}$.

L'heuristique **Random permutation** de Schalekamp [117] ressemble à Pick-a-perm mais la permutation médiane approximée est tirée aléatoirement dans toutes les permutations de S_n.

2.4.2. Heuristiques à base de système électoral

Le problème de la médiane de permutations sous la distance de Kendall-τ étant un système électoral, plusieurs autres systèmes électoraux peuvent être utilisés pour approximer

la permutation médiane. Dans ce contexte on parle souvent des "candidats" pour désigner les éléments et des "votes" ou "électeurs" pour désigner les permutations.

La méthode **Borda** ou "BordaCount" [27] consiste à ordonner les éléments selon leur position moyenne dans les permutations de l'ensemble de départ $\mathcal{A}$.

La méthode **Copeland** [45] consiste à ordonner les éléments selon leur nombre de victoires moins le nombre de défaites contre chaque autre élément. Un élément est victorieux sur un autre élément s'il est préféré à ce dernier dans la majorité des permutations de l'ensemble de départ $\mathcal{A}$. Les égalités valent 1/2 point pour chaque élément de la paire. On retrouve parfois une formulation équivalente [134] dans laquelle on ne fait qu'additionner 1 point par victoire, 1/2 par égalité et 0 par défaite puis on tri les candidats selon ce score. Une méthode similaire, ne comptant que les victoires, fut proposée par un moine (Llull) du XIII siècle [42, 86], elle sera nommée ici **CopelandWins**.

La méthode **Ranked Pairs** [130], ordonne les paires d'éléments (x,y) en fonction du nombre de votes qui préfèrent x à y, où x est le gagnant entre x et y. Un graphe orienté est ensuite construit de façon itérative en ajoutant les arêtes correspondantes au paires dans l'ordre de l'ordonnancement précédent. Si une arête à ajouter crée un cycle dans le graphe, alors cette arête n'est pas ajoutée. À la fin, un graphe orienté est obtenu (pas nécessairement complet) et l'élément gagnant est celui qui n'a aucun arc qui pointe vers lui. Une permutation médiane approximée peut alors être choisie arbitrairement parmi toutes les permutations qui respectent l'ordre partiel induit par le graphe résultant.

La méthode **Shulze** [119] se base sur le calcul du plus large chemin entre deux sommets dans un graphe orienté. Dans le cas présent, c'est le graphe de tournoi qui est utilisé où le poids de chaque arc (x,y) représente les votes préférant x à y. Pour chaque paire d'éléments, la méthode calcule le chemin le plus large (i.e. tel que tous les arcs sur ce chemin sont au moins de poids k) dans les deux sens. La "largeur" de ces chemins représente la force de préférence entre les éléments. Un graphe résultat est créé tel que chaque arc (x,y) a le poids du plus large chemin menant de x à y. Si ce graphe est un ordre total alors l'ordre topologique des sommets donne une permutation. Dans le cas où il y a un ordre qui n'est pas total (donc avec des égalités), une méthode pour casser ces égalité consiste à piger des votes aléatoirement et décider l'ordre des paires en fonction de ces vote.

Dans [112], plusieurs autres systèmes électoraux basés sur le vote d'un candidat unique sont adaptés pour gérer des votes à plusieurs candidats. La méthode **Plurality** déclare vainqueur le candidat qui a reçu le plus de votes des électeurs. Les égalités sont cassées arbitrairement. La version adaptée de Plurality considère seulement le premier candidat

de chaque permutation comme vote. Le candidat vainqueur est calculé puis rajouté dans une liste ordonnée des vainqueurs. Ce candidat est enlevé des permutations et le procédé se répète jusqu'à ce que tous les candidats soient intégrés dans la liste des vainqueurs qui donne le résultat final. La méthode **Raynaud** retire itérativement le candidat i qui a la plus grande défaite avec un autre candidat j et le rajoute dans une liste de perdants. La grandeur de la défaite est mesurée par le nombre L_{ji} (Définition 1.1.6) de votes préférant j à i. Dans la version adaptée, la méthode ne peut retirer qu'un seul candidat même si plusieurs partagent une défaite de même grandeur. La liste des perdants est renversée pour donner la permutation résultat. La méthode **Simpson-Kramer** ou méthode **minimax** déclare vainqueur le candidat dont la plus grande défaite est inférieure aux plus grandes défaites de tous les autres candidats. Dans l'adaptation, pour obtenir une permutation, les égalités sont cassées en considérant la seconde pire défaite des candidats à égalité.

2.4.3. Heuristiques à base de chaînes de Markov

Dans [56], une première heuristique basée sur les chaînes de Markov est proposée. Cette heuristique **MC4**, qui origine d'applications sur le web, fonctionne similairement à PageRank, le fameux algorithme de tri de pages web de Google [32, 106]. Chaque candidat est un état. D'un état quelconque l'heuristique peut transiter vers n'importe quel autre état avec probabilité égale, mais seulement si le candidat correspondant au second état est vaincu à majorité dans les votes par celui correspondant à l'état de départ. Il y a une probabilité de 1/7 de se téléporter dans tout autre état aléatoire. La distribution stationnaire est calculée pour ce modèle puis l'heuristique retourne un ordonnancement en ordre croissant des candidats par rapport à cette distribution. L'heuristique est calculable en temps polynomial et est une 2-approximation.

Dans [117], les auteurs utilisent la même idée que celle présentée dans [56] et proposent **MC4Approx**, une version approximée de MC4 qui est calculé plus rapidement en évitant de calculer la distribution stationnaire.

2.4.4. Heuristiques basées sur d'autres méthodes d'agrégation

La distance de "Spearman's footrule" entre deux permutations est la somme des différences (positives) de positions de tous les éléments : $d_{Sf}(\pi,\sigma) = \sum_{i\in[n]} |\pi_i^{-1} - \sigma_i^{-1}|$. Notez que cette distance est une borne supérieure à la distance de Kendall-τ. En fait, il a été montré dans [54] que $d_{Sf}(\pi,\sigma) \leq d_{K\tau}(\pi,\sigma) \leq 2 \times d_{Sf}(\pi,\sigma)$. La **Footrule aggregation** décrite par Diaconis et Graham dans [54] consiste à trouver une permutation qui minimise la somme

des distances de "Spearman's footrule" entre cette permutation consensus et les permutations de l'ensemble de départ $\mathcal{A}$. Cette permutation peut être trouvée en résolvant le problème d'assignation [56] dans lequel les éléments doivent être assigné à des positions et chaque assignation a un coût. Une solution à ce problème peut être calculée en $O(n^3)$ avec l'algorithme hongrois [83] (voir [6]).

2.4.5. Heuristiques gloutonnes

Dans [61], le problème du plus proche voisin sur une requête dans une base de données (un ensemble de points dans d dimensions) est transformé en problème d'agrégation de classements. Les auteurs montrent que la méthode de Kemeny-Young est efficace pour ce problème et l'heuristique **MEDRANK** (Median Rank Aggregation) est présentée. Cette heuristique lit en parallèle de gauche à droite les m permutations de $\mathcal{A}$, élément par élément. Avec un ratio $h \in [0,1]$ préalablement fixé, l'heuristique construit la permutation consensus en ajoutant à droite le ou les éléments qui ont été lus dans $h \times m$ permutations ou plus. Les égalités entre éléments sont cassées de manière arbitraire.

Une heuristique gloutonne **Greedy-Order** (parfois retrouvée sous le nom **CSS**) d'un facteur d'approximation d'ordre 2 a été décrite dans [39]. Cette heuristique construit une permutation en choisissant à chaque étape un prochain élément à insérer à la fin du préfixe de la permutation en construction. Le préfixe commence vide et l'ensemble V contient tous les éléments restant à placer. À chaque étape, est calculé un score pour chaque élément x de l'ensemble V. Ce score, pour x, est la somme des votes qui préfèrent x à y moins la somme des votes qui préfèrent y à x, pour chaque élément y de V. À chaque étape est choisi l'élément qui maximise ce score. Cet élément est alors retiré de l'ensemble V puis ajouté à la fin du préfixe.

Il faut noter que cette heuristique n'a pas été spécialement inventé pour le problème de la médiane de permutations mais pour le problème plus général sur les graphes. Un graphe orienté (pas nécessairement complet) avec poids est construit, où les poids indiquent la force des préférences entre des éléments. Dans ce contexte on vise à maximiser les accords (les préférences), ce qui est similaire à minimiser les désaccords. Ce problème est bien connu dans la littérature sous le nom de Linear Ordering Problem (LOP) et a été introduit en 1958 dans [38]. Depuis, de nombreux chercheurs s'y sont intéressés [34, 46, 69, 85, 90, 118]. En fait, quelques heuristiques présentées dans cette section proviennent du problème LOP dont Greedy-Order [39], présenté dans cette section et Chanas [35], présenté la Section 2.4.7.

Dans [51] est proposé une heuristique gloutonne **Greedy graph heuristic** basée sur le graphe orienté de tournoi correspondant à l'ensemble $\mathcal{A}$ (Définition 1.1.7). En premier lieu, le critère de Condorcet étendu (voir la section 2.7) est appliqué sur le graphe pour identifier des composantes connexes. En second lieu, des paires d'éléments sont ordonnées itérativement, en partant des paires dont la différence entre l'ordre majoritaire et l'ordre minoritaire est la plus élevée (voir Définition 1.1.8). La différence utilisée entre ces deux ordres compte le nombre de permutations qui ont l'ordre majoritaire pour une paire d'élément moins le nombre de permutations qui ont l'ordre minoritaire pour cette paire d'élément. À chaque itération, une fermeture transitive est appliquée sur l'ensemble des paires.

2.4.6. Heuristiques à base de tri

Plusieurs algorithmes de tri ont été testés comme heuristique sur le problème de la médiane. Ces algorithmes utilisent aussi l'idée d'ordre majoritaire entre paires d'éléments. Si $\mathcal{A}$ contient un nombre impair de permutations, il y a toujours un ordre majoritaire, sinon, il est possible qu'on ai l'ordre $i \prec j$ dans la moitié des permutations de $\mathcal{A}$ et l'ordre $j \prec i$ dans le reste. Dans ce cas, il n'y a pas d'ordre majoritaire pour la paire d'éléments (i,j).

KwikSort[3] ou **QuickSort** [135] est un algorithme de tri (basé sur le tri rapide) qui consiste à choisir un élément pivot et de classer tous les autres éléments à sa gauche ou à sa droite selon l'ordre majoritaire par rapport au pivot. Les éléments qui n'ont pas d'ordre majoritaire par rapport au pivot sont classés arbitrairement. La méthode est répétée récursivement sur les éléments à gauche du pivot et ceux à droite. Une version déterministe **DetQuickSort** [135] évalue chaque élément comme pivot potentiel et choisi celui qui minimise le nombre de désaccords engendrés par l'opération de pivot entre les éléments à gauche et ceux à droite. Une version plus rapide **LogQuickSort** [135] consiste à évaluer $log(n)$ éléments comme pivots et semble être aussi efficace [117]. La version probabiliste est d'un facteur d'approximation 11/7 et celle déterministe est d'un facteur d'approximation 8/5.

MergeSort[117], basé sur le tri fusion, "tri" les éléments en les scindant en deux listes à trier. Une fusion de deux listes consiste à créer une troisième liste en considérant le premier élément de chaque liste et en choisissant celui qui est préféré en majorité à l'autre.

InsertionSort[56, 117], la dernière heuristique à base de tri, consiste à insérer un à un les éléments dans une liste triée. L'insertion commence par placer le nouvel élément à la fin de la liste et le remonte jusqu'au premier élément qui est préféré à majorité au nouveau.

Ces trois heuristiques à base de tri ont l'inconvénient de trier localement les éléments. Dans le cas où la transitivité est respectée par la relation de majorité sur les éléments, ce n'est pas un inconvénient, mais dans la majorité des cas, la transitivité n'est pas respectée.

2.4.7. Heuristiques à base d'optimalité locale

Certains auteurs ont étudié une approche locale pour la résolution du problème. Étant donné une permutation choisie, ces approches vont localement optimiser la distance de Kendall-τ à l'ensemble de permutations $\mathcal{A} \subseteq S_n$ dans le but de se rapprocher de la médiane.

L'heuristique **2-opt** consiste à appliquer sur une permutation choisie, des échanges d'éléments adjacents jusqu'au moment où aucun tel échange ne puisse diminuer le score de Kemeny. De la même façon, **k-opt** (travail non-publié [95]) est la généralisation où k éléments consécutifs sont réarrangés pour faire descendre la distance à $\mathcal{A}$. On dira qu'une permutation est k-optimale si aucun réarrangement d'un segment de k éléments consécutifs peut descendre le score de Kemeny. Dans [56], on parle de **Local Kemenization** à propos de 2-opt et une permutation est dite "locally Kemeny optimal" si aucune transposition de paire adjacente peut réduire le score de Kemeny.

L'heuristique de **Chanas** [35] consiste, comme 2-opt, à faire des échanges d'éléments adjacents sur une permutation choisie, inverser la permutation résultat puis réappliquer le processus. Dans cette heuristique, dès qu'un échange est effectué, on recommence à vérifier du début de la permutation. Dans [41], une version améliorée de cette heuristique **ChanasBoth** ne doit plus revenir au début mais peut continuer à inverser les paires dans les prochaines positions.

2.4.8. Heuristiques à base d'approche exacte limitée ou relaxée

D'autres auteurs ont exploré des idées d'heuristiques en limitant les ressources des algorithmes de résolution exacte de type branch-and-bound.

Dans [92], Meila et al. dérivent une heuristique de recherche en faisceaux **Beam search** en limitant le nombre de branchement à chaque étape de leur algorithme branch-and-bound [92] (voir Section 2.3). Dans cette heuristique, les noeuds sont des préfixes de permutations et le branchement consiste à ajouter le prochain élément au préfixe.

LPRelaxation (ou **LP**) est la relaxation continue du modèle de programmation en nombres entiers (ILP) (voir Section 2.3). Cette borne inférieure (voir Section 2.8) est aussi utilisée comme une heuristique dans [**6, 117**]. Dans [**117**], il est montré que LPRelaxation trouve souvent le score de Kemeny d'une médiane dans le cas des données provenant de recherches sur le web (34 des 37 cas étudiés dans leur travail). C'est-à-dire que la valeur à l'optimum de la relaxation est entière. Si on arrondit les valeurs des variables d'ordre, il y a une possibilité de retrouver une permutation médiane. Il y a aussi une possibilité qu'on obtienne une solution qui ne peut pas engendrer de permutation.

Dans [**3**], Ailon et al. décrivent une méthode qui renvoie le meilleur résultat entre Pick-a-perm et une heuristique à base de relaxation du modèle ILP (Section 1.1.3). Cette dernière heuristique, nommée **LpKwikSort**, résout le problème LP (relaxé) et choisi un élément pivot p de façon aléatoire. Tous les autres éléments i sont ensuite placés soit à gauche soit à droite de façon probabiliste proportionnellement en fonction des valeurs des variables d'ordre $x_{pi} \in [0,1]$ de la solution du modèle. Une variable d'ordre $x_{pi} = 1$ indique que p précède i dans la solution.

Dans [**2**] Ailon perfectionne **LpKwikSort** (aussi appelée **Ailon 3/2**) en changeant la probabilité qu'un élément soit placé avant ou après le pivot : si un élément i a $x_{pi} < \frac{1}{6}$ alors la probabilité devient 0 alors que si un élément i a $x_{pi} > \frac{5}{6}$ alors la probabilité devient 1. Les valeurs intermédiaires sont réajustées selon une fonction linéaire. L'heuristique est décrite en détails dans la section sur les classements avec égalité (voir Section 2.10).

2.4.9. Heuristiques à base de recherche locale

Plusieurs heuristiques utilisent l'opération de replacer un élément dans la permutation. Ce type d'heuristiques est très populaire pour résoudre le problème LOP [**46, 66**]. Cette opération est connue sous les noms de BestFit [**15**], INSERT_MOVE [**66**], MOVES heuristic [**41**], Local Search [**6**], single vertex move [**80**] ou mouvement circulaire [**25**].

Un **mouvement circulaire** consiste à déplacer un segment d'éléments vers la droite ou vers la gauche d'une position circulairement i.e. en renvoyant le dernier élément du segment au début. Formellement, le mouvement circulaire $c[i,j]$ $(1 \leq i \neq j \leq n)$, on suppose $i < j$ sans perte de généralité, appliqué à une permutation $\pi \in S_n$ est défini comme suit :

$$c_r[i,j](\pi) = \pi_1 \dots \pi_{i-1} \boldsymbol{\pi_j \pi_i \dots \pi_{j-1}} \pi_{j+1} \dots \pi_n$$

pour le mouvement circulaire vers la droite et

$$c_l[i,j](\pi) = \pi_1 \dots \pi_{i-1} \boldsymbol{\pi_{i+1} \dots \pi_j \pi_i} \pi_{j+1} \dots \pi_n$$

pour le mouvement circulaire vers la gauche.

Notez que d'après des observations préliminaires, nous avons remarqué que les permutations médianes d'un ensemble de permutations sont souvent reliées par un ou plusieurs mouvements circulaires.

Dans [25] et [26], l'heuristique **FindMedian** est introduite. Cette heuristique part de chaque permutation de l'ensemble des permutations de départ et y effectue des mouvements circulaires pour diminuer le score de Kemeny. Lorsque l'heuristique ne trouve plus de mouvements circulaires avantageux, elle peut effectuer jusqu'à deux 0-mouvements, des mouvements qui ne changent pas le score de Kemeny. Cette heuristique a été testée sur des ensembles de permutations de 3 à 12 éléments, et trouve toujours la vraie médiane quand $n \leq 6$. Pour $7 \leq n \leq 12$, le pourcentage d'erreur croît de 0.05% à 1.6%.

La recherche avec voisinage variable ou **VNS** (pour "Variable Neighborhood Search") est une métaheuristique qui consiste à alterner entre la recherche locale et la diversification de la solution. Dans [15], les auteurs décrivent leur heuristique et la testent sur des données réelles et synthétiques. L'heuristique débute avec une solution (permutation) aléatoire et va itérativement effectuer un processus pour un nombre d'itérations déterminé. Le processus consiste à effectuer itérativement une étape de diversification dans laquelle la solution courante va être modifiée suivie d'une étape de recherche locale dans laquelle l'heuristique va chercher à diminuer le score de Kemeny de la solution courante. L'étape de la diversification consiste à effectuer k mouvements circulaires aléatoires sur la solution courante. Au fil des itérations du processus, k augmente (jusqu'à un maximum déterminé), permettant ainsi d'aller chercher des solutions de plus en plus éloignées. La recherche locale est une version modifiée des mouvements circulaires qui diminuent le score de Kemeny. Dans cette version, la recherche locale peut considérer parfois jusqu'à deux mouvements circulaires imbriqués. Lorsqu'une meilleure solution est trouvée, celle-ci est sauvegardée en mémoire puis le processus s'arrête, une nouvelle itération de l'heuristique commence et le processus est relancé. En bref, cette métaheuristique essaye de trouver un optimum global en cherchant dans un voisinage de plus en plus étendu d'un optimum local. Quand, un meilleur optimum local est trouvé, la métaheuristique recommence à partir de ce nouveau point avec un petit voisinage. Dans cet article, les auteurs utilisent cette métaheuristique sur des données qui incluent des classements (partiels) d'articles (ou de candidats) par des

arbitres et poursuivent avec un deuxième algorithme de recommandation qui considère aussi des scores donnés par les arbitres sur les articles. Les auteurs tentent d'estimer un score moyen pour les articles à partir de classements incomplets (chaque arbitre n'examine qu'un sous-ensemble d'articles).

Dans [8], deux heuristiques, qui utilisent une variation du mouvement circulaire et une approche gloutonne, sont décrites. **QUICK** qui se trouve à mi-chemin entre Insertion-Sort (Section 2.4.6) et LS (Section 2.4.11), part d'une permutation guide donnée par une variante de l'heuristique de Borda (Section 2.4.2). Une liste vide est créée et le premier élément de la permutation guide est inséré dans la liste. L'heuristique QUICK va ensuite prendre l'élément suivant et évaluer toutes les positions pour insérer le nouvel élément. La position qui minimise le score (partiel) sera choisie. Ce processus est répété jusqu'à ce que tous les éléments soient placés. L'heuristique va ensuite répéter le tout avec la nouvelle solution comme permutation guide. L'heuristique **FAST** exécute l'heuristique QUICK originale une première fois puis répète pour un nombre d'itérations défini (manuellement en fonction de n), l'heuristique QUICK ayant ici comme point de départ une permutation guide aléatoire. Les heuristiques ont été évaluées sur des données simulées allant jusqu'à 20 éléments et des données réelles allant jusqu'à 50 éléments.

Dans [12], plusieurs heuristiques intéressantes sont proposées avec leurs combinaisons. De plus, deux critères supplémentaires sur les permutations médianes sont énoncés. Ces critères permettent de différencier les "meilleures" médianes. Le premier critère est de choisir la ou les médianes dont la distance maximale entre la médiane et une permutation de $\mathcal{A}$ est minimale. Le second critère est de minimiser la variance des distances entre une médiane et les permutations de $\mathcal{A}$. Ces deux critères poussent les désaccords entre les permutations de $\mathcal{A}$ à être distribués plus équitablement. L'heuristique **Subiterative Convergence** ou **SC** consiste à optimiser à l'intérieur d'une fenêtre de longueur η, avec typiquement $\eta \in \{2,...,8\}$. Étant donné un ensemble de permutations $\mathcal{A} \subseteq S_n$ et une permutation de départ π, l'heuristique regarde les η premiers éléments de π et optimise sur ce segment. Le premier élément de ce segment est placé au début d'une liste vide et l'heuristique recommence en déplaçant la fenêtre de taille η d'une position vers la droite. Le processus se termine lorsque la nouvelle liste contient n éléments, ce qui termine la première itération. Le dernier segment optimisé est ajouté en entier dans la liste. L'heuristique répète itérativement le processus jusqu'à convergence du résultat. Les auteurs utilisent l'heuristique de Borda (Section 2.4.2) pour obtenir la première permutation. Ils

décrivent ensuite une stratégie pour répéter l'heuristique SC sur des permutations différentes. L'heuristique **Greedy Algorithm** ou **GA** est la même que LocalSearch (Section 2.4.9) avec un paramètre s, $s \leq 30$ recommandé, qui limite le déplacement d'un d'élément vers une nouvelle position. Tous les éléments ne sont considérés qu'une seule fois. Les heuristiques **FUR** et **SIgFUR** sont des méthodes un peu plus élaborées, basée sur Subiterative Convergence et Greedy Algorithm. FUR répète Subiterative Convergence et Greedy Algorithm jusqu'à convergence du résultat avec plusieurs valeurs de η. SIgFUR consiste à appliquer la méthode répétée de Subiterative Convergence puis d'appliquer Greedy Algorithm et finalement FUR. Les heuristiques sont testées et comparées contre QUICK [8] et FAST [8] sur des ensembles de permutations allant jusqu'à $n = 400$. Il en sort que les heuristiques FUR et SIgFUR donnent des meilleures approximations que QUICK et FAST.

2.4.10. Heuristiques à base d'évolution

Les métaheuristiques à base d'évolution [124], ou algorithmes évolutionnistes, sont des méthodes basées sur les processus d'évolution et de sélection naturelle observées en biologie. Dans ce contexte, une solution à un problème d'optimisation peut être vue comme un individu d'une espèce. L'algorithme gère une population d'individus qui évolue à travers les générations. La pression sélective de l'évolution, implémentée par la sélection d'individus plus optimaux, va faire converger la population vers des optimums du problème.

Dans [4], Aledo et al. présentent l'heuristique **Genetic Algorithm** ou **GA**, une implémentation de la métaheuristique portant le même nom (algorithme génétique). GA commence avec une population de permutations aléatoires et va itérativement créer des générations à partir des générations précédentes. Pour créer la population de la génération suivante G_{i+1}, GA va effectuer des croisements entre les permutations de la génération G_i, puis va effectuer des mutations sur ces descendants. Une étape de sélection va choisir les meilleures permutations tirées des descendants et des parents pour peupler la nouvelle génération. Le processus s'arrête quand il n'y a eu aucune amélioration du meilleur score depuis plusieurs générations. Les opérations de croisements sont : POS, où les éléments de positions sélectionnées d'un parent sont fixés et les éléments restant sont réordonnés en fonction de l'autre parent, OX1, où un segment de la permutation d'un parent est fixé et les éléments restants sont positionnés en fonction de l'autre parent (dans l'ordre de lecture à partir de la fin du segment) dans le reste de la permutation et OX2 qui ressemble à POS mais dans laquelle les positions sélectionnées diffèrent entre les parents. Les opérations de

mutations sont : l'insertion d'un élément, le déplacement d'un segment et le déplacement d'un segment inversé. Les auteurs comparent leur méthode avec CSS [**39**] (Section 2.4.5), Borda [**27**] (Section 2.4.2), Greedy Graph heuristic [**51**] (Section 2.4.5) et Beam Search [**92**] (Section 2.4.8) sur des ensembles de permutations de taille allant jusqu'à $n = 250$. Il résulte que GA donne des meilleures approximations que les quatre autres méthodes. Par contre, le temps d'exécution est significativement plus grand que celui de Beam Search qui est déjà la plus lente de ces méthodes. C'est donc une méthode qui compromet la vitesse pour avoir une meilleure précision.

Dans [**49**], d'Ambrosio et al. proposent une heuristique d'évolution différentielle (DE) nommée **DECoR** pour *Differential Evolution algorithm for the Consensus Ranking* pour le problème de la médiane. Les individus de la première génération sont des permutations aléatoires et ceux de chaque nouvelle génération sont générés à partir des individus de la dernière génération à l'aide de mutations et croisements. Ceux-ci visent à explorer l'espace de recherche mais aussi à garder les caractéristiques des bonnes solutions. La "fitness" d'un individu est associée à son score de Kemeny. Alors que DE a été conçu pour des problèmes d'optimisation continue, les auteurs contournent le problème avec une astuce nommée "approche hiérarchique" dans laquelle un vecteur de n valeurs représente un individu. Pour trouver la permutation correspondante, il suffit d'ordonner les positions en fonction de leur valeur. La mutation se fait en tirant aléatoirement trois individus : soient v_1, v_2 et v_3 les trois vecteurs correspondants aux trois individus, le vecteur du nouvel individu i est $v_i = v_1 + F \times (v_2 - v_3)$, où $F \in [0,2]$. Le croisement se fait en prenant les valeurs soit de v_i ou soit de x_i, l'individu i de la dernière génération. Le résultat est comparé à l'individu x_i et le meilleur des deux est sélectionné comme individu i de la nouvelle génération. Un paramètre $\#P \in \{10,...,30\}$ définit la taille de la population. Un paramètre L fixe le nombre maximale de générations sans amélioration après quoi, l'algorithme s'arrête. L'heuristique DECoR est comparée favorablement contre plusieurs autres heuristiques dont CSS [**39**] (Section 2.4.5), QUICK [**8**] (Section 2.4.9), GA [**4**] (Section 2.4.10), Greedy Graph heuristic [**51**] (Section 2.4.5), LS 2.4.11 et un recuit simulé (SA) de base sur des données allant jusqu'à $n = 200$.

Dans [**5**], Aledo et al. discutent de l'avantage d'utiliser des méthodes plus rapides et optimisées pour évaluer le score de Kemeny (et autres scores) d'une nouvelle permutation à partir d'une permutation au score connu. Ceci est particulièrement d'intérêt dans les métaheuristiques à base d'évolution dans lesquelles de nombreuses opérations et évaluations de score sont exécutées sur des permutations. Sachant que le recalcul du score de

Kemeny est souvent la partie la plus coûteuse en temps d'une métaheuristique à base d'évolution, les auteurs proposent l'évaluation partielle dans laquelle l'heuristique peut interrompre l'évaluation si on peut déjà déterminer que la nouvelle solution ne sera pas acceptée par l'heuristique. Les auteurs montrent l'avantage de leurs idées sur des données réelles et simulées en testant trois types d'opérations : l'insertion, l'inversion d'un segment et la transposition adjacente.

2.4.11. Heuristiques de raffinement

Une catégorie différente d'heuristiques est celle des heuristiques de raffinement. Les idées sont très similaires à celles des heuristiques présentées précédemment mais consistent en des opérations appliquées aux permutations résultantes de ces heuristiques.

L'heuristique **LocalSearch** ou **LS** [**6, 41, 80, 117**] est similaire à FindMedian [**25**] (Section 2.4.9). Elle consiste à déplacer un élément de la permutation à une nouvelle position qui diminue le score de Kemeny. Ce processus est itéré jusqu'à ce qu'aucun déplacement d'aucun élément ne puisse diminuer le score. L'heuristique LocalSearch part d'une permutation donnée en entrée provenant d'une autre heuristique. Dans ce sens, l'heuristique FindMedian peut être vue comme LocalSearch appliquée au résultat de Pick-a-perm (répétée pour chaque permutation de l'ensemble de départ). La différence est que FindMedian permet des déplacements d'éléments qui gardent le même score de Kemeny alors que LocalSearch permet seulement les déplacements qui diminuent ce score. L'heuristique LocalSearch peut être implémentée soit pour effectuer le premier mouvement circulaire évalué qui diminue le score de Kemeny ou pour évaluer tous les mouvements circulaires et effectuer celui qui diminue le plus le score. Dans [**80**], une permutation est dite localement optimale si aucun mouvement circulaire ne peut être exécuté pour diminuer le score de Kemeny.

Les méthodes **IS**, **MS**, **KS** [**6, 117**] qui tirent leurs noms des méthodes InsertionSort, MergeSort et QuickSort consistent à appliquer ces méthodes inspirées du tri en donnant une approximation de la médiane comme entrée à ces méthodes. IS va insérer les éléments dans l'ordre donné par l'approximation. MS et KS vont trier le tableau des éléments donné par l'approximation. À proprement parler, IS, MS et KS ne garantissent pas une diminution du score de Kemeny donc ne sont pas un raffinement de la solution dans tous les cas. En pratique, ces raffinements sont appliquées à des heuristiques très rapides et peu précises comme Borda et Copeland (voir Section 2.4.2).

Les méthodes **2-opt**, **3-opt** et **k-opt** [95] consistent à parcourir une approximation de gauche à droite en considérant une fenêtre de taille 2, 3 ou k respectivement et en optimisant à l'intérieur de cette fenêtre.

Comme vous venez de le voir, plusieurs approches heuristiques ont été explorées. L'avantage commun de ces heuristiques est la vitesse de calcul : des approximations sont obtenues assez rapidement avec la majorité des heuristiques. L'inconvénient majeur est la précision de ces méthodes : les facteurs d'approximation sont assez élevés. Une approche pertinente à explorer serait alors de concevoir des heuristiques qui trouvent une permutation médiane avec une bonne probabilité. Dans la section suivante, un schéma d'approximation en temps polynomial a été développé pour pouvoir approximer une médiane avec une précision désirée.

2.5. Schéma d'approximation en temps polynomial

Un schéma d'approximation en temps polynomial (PTAS) est donné dans [80]. C'est un algorithme qui donne une approximation de facteur $(1+\epsilon)OPT$ où OPT est le score de Kemeny d'une médiane dans un temps qui dépend de ϵ. Pour une instance du problème, i.e. un ensemble de permutations $\mathcal{A} \subseteq S_n$, le temps d'exécution est de l'ordre de :

$$O(n^3 log n(log(1/b) + 1/\epsilon)) + n2^{\tilde{O}(1/(\epsilon b)^6)},$$

où $\tilde{O}$ désigne la complexité en négligeant les facteurs logarithmiques, pour une version non-déterministe, b étant une constante fixée par l'algorithme. Le temps est exponentiel en $1/\epsilon$, ce qui n'est pas très pratique et aucune application n'est faite sur des données réelles ou aléatoires. Ce travail est une première en PTAS pour ce problème et réduit considérablement le facteur d'approximation (qui était constant avec les heuristiques mentionnées ci-haut) mais est d'intérêt théorique seulement [3, 19, 123, 135].

2.6. Algorithmes à paramètres fixes et kernelisation

Les algorithmes à paramètres fixes ("Fixed Parameter Tractability" ou FPT) [55] sont une approche en informatique pour attaquer des problèmes difficiles à résoudre en trouvant des paramètres qui peuvent en cerner la difficulté. Dû au fait que le problème de la médiane est NP-Difficile, il est impossible d'échapper à la croissance exponentielle du temps de calcul en fonction de la taille d'entrée mais le facteur de l'exponentielle peut être un paramètre qui, lorsque petit, rend le temps de calcul raisonnable. On dit qu'un problème

est FPT s'il peut être décidé en temps $f(k)\times x^{O(1)}$ où $x^{O(1)}$ est un polynôme en fonction de la taille d'entrée et $f(k)$ est une fonction exponentielle sur un paramètre k de l'instance.

Un concept proche des algorithmes FPT est celui de la kernelisation qui permet de réduire le problème à une instance plus petite (le noyau - "kernel") et plus facile à résoudre. Cette réduction doit être exécutée en temps polynomial et la taille du noyau est souvent en fonction de paramètres du problème.

De plus, plusieurs travaux ont suggéré un lien empirique entre des paramètres de similarité et la difficulté de résolution du problème (voir section 2.4). La complexité paramétrée a alors été étudiée dans une série de plusieurs articles récents. Les tableaux 2. I et 2. II font le tour des paramètres investigués puis des résultats de FPT et de kernelisation.

Notation	Paramètre
n	nombre d'éléments dans une permutation
m	nombre de permutations dans $\mathcal{A}$
k	score de Kemeny d'une médiane
d	distance maximale entre 2 permutations de $\mathcal{A}$
d_a	distance moyenne entre les permutations de $\mathcal{A}$
r_{max}	étendue maximale de positions pour un élément

TABLEAU 2. I. Paramètres du problème de la médiane de permutations et descriptions.

Premièrement, dans [19][21] Betzler et al. proposent trois algorithmes : un qui dépend du score de Kemeny k d'une médiane, le deuxième qui dépend de la distance maximale d entre deux permutations de l'ensemble de départ $\mathcal{A}$ et le troisième qui dépend du nombre n d'éléments dans chacune des permutations. Dans ce travail, une "dirty pair" est une paire d'éléments telle qu'il existe au moins deux permutations dans $\mathcal{A}$: une avec $i \prec j$ et l'autre avec $j \prec i$, c'est-à-dire qui n'a pas unanimité dans les votes. Les auteurs décrivent une réduction de l'instance (kernelisation) en temps polynomial à une instance d'au plus $2k$ permutations et $2k$ éléments en enlevant les éléments qui ne font partie d'aucune "dirty pair" et en vérifiant s'il y a plus de k votes identiques pour des sous-ensembles d'éléments. Un algorithme BnB est décrit qui branche en fixant l'ordre relatif de triplets d'éléments. La complexité de ce BnB est de l'ordre de $O(1.53^k + n^2m)$ où n^2m vient de la réduction faite en prétraitement. Un deuxième algorithme de programmation dynamique est paramétré par d, la distance maximale entre 2 permutations de $\mathcal{A}$, et se calcule en $O((3d+1)!\times dlogd\times nm)$. L'algorithme considère des segments de longueur $d+1$ parallèlement dans toutes les permutations de $\mathcal{A}$ et construit un consensus en optimisant à l'intérieur de ce segment. Ils démontrent que ces segments à optimiser contiennent au

Auteurs	Année	Résultats FPT
Betzler et al.[19][21]	2008	$O(1.53^k + n^2m)$
		$O((3d+1)! \times dlogd \times nm)$
		$O(2^n \times n^2m)$
Betzler et al.[20]	2009	$O(16^{d_a} \times (d_a^2 \times n + d_a \times n^2 \times logn \times m))$
		$O(32^{r_{max}} \times (r_{max}^2 m + r_{max} n^2 m logn) + m^2 nlogn)$
Simjour [123]	2009	$O(5.823^{\frac{k}{m}})$
		$O(1.403^k)$
		$O(4.829^d)$
		$O(36^{\frac{k}{m}})$*
Karpinski et Schudy [75]	2010	$O(2^{O(\sqrt{\frac{k}{m}})} + n^{O(1)})$
Fernau et al. [63]	2011	$O(2^{O(\sqrt{k}\log k)})$
Betzler et al. [22]	2011	kernelisation, nouvelle taille : $162d_a^2 + 9d_a$
Nishimura et Simjour[104]	2013	$O(4^{\frac{k}{m}})$*
Betzler et al. [18]	2014	kernelisation, nouvelle taille : $\frac{16}{3}d_a$

TABLEAU 2. II. Récapitulatif des travaux FPT. L'étoile (*) indique que l'algorithme permet d'énumérer toutes les médianes d'un ensemble.

plus $3d+1$ éléments. Finalement, le troisième algorithme calculable en $O(2^n \times n^2 m)$ se base sur le fait que pour trouver une médiane d'un sous-ensemble d'éléments C' (par projection sur $\mathcal{A}$), il suffit de considérer chaque sous-ensemble C'' construit à partir de C' en retirant un élément e. Pour chaque sous-ensemble C'', on calcule la médiane $\pi^* = \pi_1^* \pi_2^* ... \pi_{\#C''}^*$ (par projection sur $\mathcal{A}$) puis on évalue le score de Kemeny de la permutation γ commençant par e suivi d'une médiane de C'' : $\gamma = e\pi_1^* \pi_2^* ... \pi_{\#C''}^*$. On sait qu'une de ces permutations sera la médiane de C' car on considère chaque élément $e \in C'$ comme premier élément de la permutation et le reste de la permutation doit être localement stable (voir Section 2.1.2.2). Le processus est appliqué de façon récursive pour trouver la médiane de chaque sous-ensemble d'éléments.

Dans [20], Betzler et al. améliorent les résultats précédents en proposant deux nouveaux algorithmes. Le premier est paramétré par la distance moyenne d_a entre les permutations de départ et se calcule en $O(16^{d_a} \times (d_a^2 \times m + d_a \times m^2 \times logm \times n))$. Le deuxième est paramétré par l'écart maximal r_{max} entre les positions d'un même élément dans deux permutations différentes de $\mathcal{A}$ et se calcule en $O(32^{r_{max}} \times (r_{max}^2 m + r_{max} m^2))$.

Dans [123], Simjour a utilisé le paramètre $\frac{k}{m}$ (distance médiane divisée par le nombre de permutations) pour obtenir un algorithme en $O(5.823^{\frac{k}{m}})$. Il a aussi obtenu des algorithmes en $O(1.403^k)$ et $O(4.829^d)$ puis un algorithme qui énumère toutes les médianes en $O(36^{\frac{k}{m}})$.

Dans [104], un algorithme en $O(4^{\frac{k}{m}})$ est proposé par Nishimura et Simjour qui permet l'énumération des permutations médianes.

Deux résultats FPT en temps "subexponentiel" ont été développé presque simultanément en 2010-2011. Dans [75], Karpinski et Schudy ont obtenu un algorithme en $O(2^{O(\sqrt{\frac{k}{m}})} + n^{O(1)})$. Dans [63], Fernau et al. obtiennent un algorithme en $O(2^{O(\sqrt{k}\log k)})$.

Dans [22], Betzler et al. introduisent un premier concept de kernelisation paramétrée par la distance moyenne. La nouvelle instance du problème (noyau) a $162d_a^2+9d_a$ candidats.

Dans [18], Betzler et al. améliorent leur kernelisation avec $\frac{16}{3}d_a$ candidats au noyau. Ce résultat est accompagné d'une réduction d'espace (Voir Section 2.7).

Dans cette suite de travaux, l'aspect de la proximité entre les permutations de l'ensemble de départ a été exploré à travers plusieurs paramètres. Par contre, il n'y a pas de résultats expérimentaux liés à ces algorithmes à paramètres fixes. La complexité paramétrée pour ce problème reste assez théorique et en pratique, la réduction de l'espace de recherche présenté à la section suivante va offrir un avantage réel pour résoudre le problème.

2.7. Réduction de l'espace de recherche pour la médiane

La réduction de l'espace de recherche est une méthode de prétraitement en temps polynomial qui sert à résoudre partiellement un problème pour réduire le temps d'exécution d'une approche exacte. Dans le cas de notre problème, une façon de réduire l'espace de recherche est de donner des contraintes sur les ordres relatifs des éléments dans les permutations médianes. En recherche opérationnelle, on appelle cette technique la fixation de variables.

Une extension du critère de Condorcet est présenté dans [133]. Le **critère de Condorcet** [43] dit que si un élément est préféré à majorité à chacun des autres éléments alors il doit être l'élément gagnant (le gagnant de Cordorcet), i.e. l'élément en première position d'une permutation médiane. La méthode Kemeny est dite de Condorcet car elle respecte ce critère. L'**extension du critère de Condorcet** ou **ECC** dit que si l'ensemble des éléments peut être séparé en deux groupes tels que chaque élément du premier groupe est préféré à majorité à chaque élément du second groupe alors un consensus de Kemeny, i.e.

une médiane, va ordonner tous les éléments du premier groupe avant ceux du deuxième groupe. Cela permet de calculer indépendamment et parallèlement le résultat pour chacun des deux groupes (ou plus si appliqué récursivement).

Dans [**26**], Blin et al. proposent une réduction d'espace basée sur deux théorèmes. Dans le premier, ils montrent que deux éléments adjacents dans une permutation médiane seront placé dans leur ordre majoritaire c'est-à-dire dans l'ordre dans lequel ils se trouvent en majorité dans les permutations de l'ensemble $\mathcal{A}$. Dans le second théorème, ils prouvent qu'une paire d'éléments qui se trouvent toujours dans le même ordre dans les permutations de l'ensemble $\mathcal{A}$ conservera cet ordre dans une permutation médiane. Ce deuxième théorème sera appelé dans ce qui suit le **Always Theorem**. Le Always Theorem est équivalent au critère de Pareto (voir Section 2.1). Notez qu'une paire d'éléments ordonnée par le Always Theorem n'est pas nécessairement ordonnée par le ECC, la paire (a,b) de l'ensemble $\mathcal{A} = \{[a,b,c,d],[a,b,c,d],[d,a,b,c],[c,d,a,b],[c,d,a,b]\}$ en est un contre-exemple.

Dans [**18**], Betzler et al. proposent une réduction d'espace basée sur des éléments pivots qui permettent de scinder le problème en deux sous-problèmes. Dans leur travail, une paire d'éléments (i,j), $1 \leq i \neq j \leq n$ est dite "non-dirty" si le nombre de permutations de l'ensemble de départ $\mathcal{A} \subseteq S_n$ appuyant un des deux ordres ($i \prec j$ ou $j \prec i$) dépasse une certaine proportion. Entre autres, ils investiguent le seuil de 0.75, et donc, si dans au moins le $\frac{3}{4}$ des permutations de l'ensemble de départ $\mathcal{A}$ l'ordre entre i et j est le même, la paire est appelée une "non-dirty pair". Si un élément forme une "non-dirty pair" avec tous les autres éléments, il est appelé un "non-dirty candidate". Avec le seuil de $\frac{3}{4}$, Betzler et al. démontrent alors que cet élément "non-dirty" va diviser les éléments d'une permutation médiane en 2 groupes : ceux qui sont préférés à lui dans la majorité des permutations de $\mathcal{A}$ vont être placé avant lui et les autres vont être placés après lui. Cette méthode sera appelée dans ce qui suit la **3/4-Majority Rule**. Comme ECC, cette réduction d'espace permet de paralléliser le calcul pour une instance.

Dans [**96**], travail réalisé pendant ma maîtrise, est proposée une réduction d'espace qui répond aux faiblesses des deux réductions d'espace précédentes : 3/4-Majority Rule et Always Theorem. La réduction est moins restrictive que la 3/4-Majority Rule et englobe la réduction d'espace du Always Theorem dans le sens où chacune des paire d'éléments ordonnée par le Always Theorem l'est aussi par notre méthode.

Elle se base sur le fait que si deux éléments ont un ordre majoritaire dans les permutations de $\mathcal{A}$ et qu'ils sont proches (en terme de positions dans les permutations) alors cet

ordre majoritaire se retrouvera dans les permutations médianes. Plus précisément, s'il y a peu d'éléments placés entre eux dans les permutations de $\mathcal{A}$ dans lesquelles ils sont dans l'ordre minoritaire, alors cette paire d'éléments va conserver son ordre majoritaire dans les permutations médianes. Cette méthode sera référée comme le **Major Order Theorem** (MOT).

Dans une version allongée de l'article, acceptée au Journal of Discrete Applied Mathematics [98], on propose une version calculable très rapidement avec une approche vectorielle. De plus, le cas où l'ordre majoritaire n'existe pas (les deux ordres sont égaux) est investigué ce qui améliore considérablement la réduction d'espace.

Dans cette section, des principes de réduction de l'espace de recherche de par la recherche de contraintes ont été présentées au lecteur. Une grande difficulté est rencontrée sur les ensembles de permutations aléatoires qui contiennent des permutations ayant peu de similitudes entre elles. Pour autant, ces méthodes semblent faire partie de la solution gagnante, lorsque combinées avec les meilleurs algorithmes et heuristiques, pour résoudre le problème de la médiane de permutations. La prochaine section introduit le travail sur les bornes inférieures qui sont très utiles dans les algorithmes exactes de type branch-and-bound.

2.8. Borne inférieure sur le score de Kemeny d'une médiane

Une borne inférieure de notre problème correspond à une valeur calculable telle que le score de Kemeny d'une médiane ne peut pas être inférieure à celle-ci. Une borne inférieure donne une information pertinente qui peut être utilisée pour terminer une heuristique d'approximation si les valeurs sont égales, pour faire de l'élagage dans un algorithme branch-and-bound et pour trouver des contraintes sur le problème. L'objectif est de trouver des bornes inférieures qui sont facile à calculer et qui sont les plus strictes possibles.

Dans [51], Davenport et Kalagnanam ont proposé une première borne inférieure intuitive pour la distance de Kendall-τ d'une médiane π^* d'un ensemble de permutations $\mathcal{A} \subseteq S_n$. Elle consiste à comptabiliser, pour chaque paire d'éléments, le nombre de permutations dans lesquelles l'ordre est minoritaire pour cette paire. Cela est fait en supposant que chaque paire pourrait être ordonnée dans l'ordre majoritaire dans une supposée médiane (cet ordonnancement fictif serait donc en désaccord avec toutes les paires d'éléments

dans les permutations dans lesquelles elles sont dans un ordre minoritaire) :

$$LowerBound_0 = \sum_{\substack{i<j \\ i,j\in[n]}} min\{R_{ij}(\mathcal{A}),R_{ji}(\mathcal{A})\},$$

où $R_{ij}(\mathcal{A})$ est la matrice droite définit à la Section 1.1 (Définition 1.1.6).

Une seconde borne inférieure est obtenue en augmentant la première borne de la manière suivante. On considère le graphe orienté $G = (V,E)$ avec poids, où chaque sommet $v \in V$ est un élément de $\{1,...,n\}$ et où E est composé de deux arcs pour chaque paire de sommets i,j : e_{ij} et e_{ji} avec les poids respectifs $w(e_{ij}) = R_{ji}(\mathcal{A})$ et $w(e_{ji}) = R_{ij}(\mathcal{A})$. Ce graphe est une représentation de l'ensemble des permutations de $\mathcal{A}$.

Un ordonnancement linéaire des sommets est possible si le graphe est acyclique. Dans ce cas il suffit de prendre l'ordre naturel des sommets induits par le graphe orienté et le résultat est une médiane. Dans le cas où il y a des cycles, le problème de la médiane revient à enlever un ensemble d'arcs d'un poids minimal pour rendre le graphe acyclique (minimum weight feedback arc set problem [74]).

Soit G' le graphe obtenu de G en annulant, pour chaque paire de sommets, les poids des arcs opposés dans les permutations de départ de $\mathcal{A}$: $w(e'_{ij}) = w(e_{ij}) - min\{R_{ij}(\mathcal{A}),R_{ji}(\mathcal{A})\}$, $\forall i,j, 1 \leq i < j \leq n$. La création de ce graphe G' se fait parallèlement avec le calcul de la première borne inférieure $LowerBound_0$.

Soit $DC_{G'}$ n'importe quel ensemble de cycles orientés disjoints du graphe G'. Le poids de l'arc de poids minimal dans chaque cycle de $DC_{G'}$ peut être ajouté à la première borne inférieure car dans un ordonnancement des éléments de $[n]$ d'une permutation médiane, il y aura au moins un arc par cycle qui ne pourra pas être satisfait :

$$LowerBound_1 = LowerBound_0 + \sum_{c\in DC_{G'}} \min_{e\in c}\{w(e)\}.$$

Pour des raisons d'efficacité de calcul, ils proposent de n'utiliser que des cycles disjoints de longueur 3.

Dans [44], Conitzer, Davenport et Kalagnanam reprennent ces travaux et poussent plus loin la borne inférieure en démontrant que les cycles ne doivent pas être nécessairement disjoints.

Soit $JC_{G'}$ une séquence $c_1, c_2, ..., c_l$ de cycles de G'. Soit $I(c,e)$ la fonction indicatrice qui dénote si l'arc e est présent dans le cycle c, *i.e.* $I(c,e) = 1$ si $e \in c$ et 0 sinon. Soit $v_i = \min_{e\in c_i}\{w(e) - \sum_{j=1}^{i-1} I(c_j,e)v_j\}$, le poids retiré de G' par chaque cycle c_i. On obtient la nouvelle borne inférieure suivante :

$$LowerBound_2 = LowerBound_0 + \sum_{i=1}^{l} v_i.$$

Des cycles de longueur 3 peuvent aussi être utilisés pour des raisons de vitesse de calcul. On peut noter que le nombre maximal de cycles de longueur 3 dans un graphe de tournoi de n sommets est donné par la formule suivante [**16**] :

$$max3cycles(n) = \begin{cases} \frac{1}{24}n(n+1)(n-1) & \text{if } n \text{ impair} \\ \frac{1}{24}n(n+2)(n-2) & \text{if } n \text{ pair .} \end{cases}$$

Finalement, Conitzer et al. [**44**] proposent une borne inférieure "**LPRelaxation**" basée sur la programmation linéaire simplement en enlevant la contrainte d'intégralité de la modélisation en programmation en nombres entiers du problème et testent cette borne avec CPLEX. Le résultat est concluant : c'est la borne inférieure la plus stricte. Cette borne inférieure est aussi utilisée par plusieurs autres auteurs [**6, 117**] comme une heuristique (voir Section 2.4). En pratique, sur de nombreuses instances tirées de données réelles, LPRelaxation (ou simplement **LP**) donne la réponse optimale au problème [**117**].

2.9. Travaux de comparaison

Seulement deux études comparatives ont été faites sur le problème, une en 2009 et l'autre en 2012. Elles comparent différents algorithmes exactes et heuristiques sur des données réelles et artificielles.

Une première étude comparative a été faite en 2009 par Schalekamp et Zuylen dans [**117**]. Plusieurs heuristiques et diverses combinaisons de celles-ci y sont évaluées sur des données provenant de recherches sur le web. L'étude recommande l'utilisation des heuristiques Borda [**27**] et Copeland [**45**] (Section 2.4.2) combinées ou pas avec IS (Section 2.4.11) pour obtenir une approximation de la médiane. Cette recommandation est basée sur le temps de calcul, la facilité d'implémentation et la qualité de l'approximation. Les heuristiques QuickSort [**3**] (Section 2.4.6) et MC4Approx (Section 2.4.3) sont aussi bien classées. Les auteurs recommandent l'utilisation de la recherche locale pour améliorer la qualité des solutions données par les heuristiques au prix d'un temps supplémentaire de calcul. Ce raffinement a permis avec leurs données de se rapprocher en moyenne à 0.03% de la solution optimale. Les auteurs montrent aussi que la solution obtenue par LPRelaxation [**44**] (Section 2.4.8) est souvent une solution optimale et, dans le peu de cas où elle ne l'est pas, elle donne une borne inférieure très serrée. Les implémentations ont été

faites sur MATLAB [1]. Les solutions optimales ont été calculées avec la programmation en nombres entiers et CPLEX [2].

Dans [6], Ali et Meila font un grand travail de comparaison entre plusieurs heuristiques et algorithmes qui résolvent le problème de la médiane de permutations. Ils y expliquent les diverses méthodes, introduisent plusieurs ensembles de données (réelles et synthétiques) et conduisent des expériences comparant ces méthodes. Ils y introduisent aussi des notions de difficulté sur les instances. Un critère en particulier est utilisé pour déterminer le "consensus" - la similarité des permutations - d'un ensemble $\mathcal{A}$: h_{borda}/h_{low} qui est le rapport entre l'approximation de la distance médiane donnée par l'heuristique Borda [27] (Section 2.4.2) et la borne inférieure triviale (voir $LowerBound_0$ à la Section 2.8). Pour résoudre le problème, le solveur ILOG CPLEX d'IBM ressort gagnant de leurs expériences. Borda [27] et Copeland [45] (Section 2.4.2), deux heuristiques simples et rapides, sont conseillées pour plusieurs cas. De plus, un algorithme de recherche locale est encore conseillé pour augmenter la qualité de la solution. Une suggestion de solution de rechange pour le solveur CPLEX est leur solveur branch-and-bound. L'heuristique Beam Search [92] (Section 2.4.8) semble bien performer alors que lorsqu'elle est couplée à la recherche locale, elle devient la meilleure heuristique selon eux. Le travail recommande d'investiguer la réduction d'espace (en prétraitement) en combinaison avec les algorithmes et heuristiques discutées.

Les tableaux suivants donnent un récapitulatif des heuristiques et des algorithmes exactes comparés dans ces deux travaux [6, 117] : Le Tableau 2. III liste toutes les heuristiques connues pour le problème de la médiane de permutations, le Tableau 2. IV liste toutes les heuristiques de raffinement connues pour ce problème et finalement, le Tableau 2. V liste tous les algorithmes exactes connus pour ce problème.

1. https://www.mathworks.com/products/matlab.html
2. https://www.ibm.com/ca-en/marketplace/ibm-ilog-cplex

2.10. Revue de littérature sur les classements

La considération des égalités dans les classements n'est pas récente. En effet, en 1945 Kendall en parle déjà dans [79]. Il propose de caractériser les classements qui contiennent des éléments à égalité en assignant pour chaque élément un rang qui est la moyenne des positions des éléments avec qui il est à égalité. Par exemple, pour le classement [[1,3],[5],[6,7,8],[4],[2]] les positions littérales (de gauche à droite) sont [[1,2],[3],[4,5,6],[7],[8]]. Quand la moyenne est appliquée pour chaque ensemble d'éléments à égalité, on a les rangs caractéristiques suivantes : [[$1\frac{1}{2}$,$1\frac{1}{2}$],[3],[5,5,5],[7],[8]]. Alors le rang caractéristique de l'élément 3 est $1\frac{1}{2}$, celui de l'élément 8 est 5 alors que celui de l'élément 2 est 8. Kendall poursuit avec des mesures de corrélation entre des classements en utilisant cette caractérisation des classements.

Ce n'est que quelques décennies plus tard que le sujet est repris et que la distance de Kendall-τ généralisée, qu'on utilise maintenant, fait son apparition.

Dans [56], Dwork et al. traitent les classements incomplets sans égalités qu'ils nomment "listes partielles"(*partial lists*). Il généralisent le score de Kemeny en utilisant la projection pour calculer la distance de Kendall-τ entre deux listes partielles qui ne partagent pas les mêmes éléments. Cette mesure est nommée distance de Kendall-τ induite (*induced Kendall-tau distance*) même si elle n'est pas une mesure de distance. La distance de Kendall-τ induite se calcule entre un classement complet sans égalité T et un classement R sans égalité. Dans la première étape on effectue la projection $T_{|R}$ de T sur l'ensemble des éléments de R. Dans la deuxième étape on calcule la distance de Kendall-τ habituelle $d_{KT}(T_{|R},R)$ où $T_{|R}$ et R peuvent être considérés comme des permutations car ils partagent les mêmes éléments et ne contiennent pas d'égalité.

Dans l'article [59] de Fagin et al., une première étude est réalisée sur les classements et la généralisation du consensus de Kemeny. Dans ce travail très théorique, la distance de Kendall-τ généralisée est introduite dans laquelle une pénalité $p = \frac{1}{2}$ est donné pour une égalité non-respectée. Une généralisation de la distance de Spearman [54] est aussi présenté où entre deux classements, on additionne pour chaque élément les différences de positions des paniers qui les contiennent. Par la suite, la métrique de Hausdorff [47] est appliquée à ces deux distances et les liens entre ces deux métriques sont discutés. Il est montré que ces deux métriques sont de la même classe d'équivalence et qu'une solution minimisante pour une est une approximation d'un facteur constant pour l'autre. Ainsi, les deux métriques se partagent les mêmes heuristiques comme approximation de facteur constant. Finalement, un algorithme d'approximation basé sur la programmation

dynamique, et sur les positions des éléments **FaginDyn**, est décrit pour minimiser ces métriques prenant seulement des classements complets en entrée.

Dans [2], Ailon traite des classements partiels avec égalités. Il s'intéresse particulièrement aux *top-m lists* et aux *p-rating*. Les *top-m lists* sont des classements incomplets qui ne contiennent pas d'égalité. Les éléments retrouvés ordonnés dans ces classements sont les m éléments de l'univers U jugés les plus importants selon une méthode de classement, d'où le nom de *top-m lists*. Un exemple d'application serait la liste des m sites web données par les k premières pages résultats d'un moteur de recherche. Les autres sites web, qui n'apparaissent pas sur cette liste sont jugés moins pertinents. Les *p-rating* sont des classements incomplets pouvant contenir des égalités. Un exemple d'application serait un classement ne contenant que les objets (restaurants, parcs, etc.) ayant une mention de 3, 4 ou 5 étoiles. Cela donne un classement de 3 paniers. Les autres objets sont jugés non intéressants. La mesure introduite est une autre généralisation de la distance de Kendall-τ qui compte entre deux classements r_1 et r_2 le nombre de paires d'éléments u et v telles que $u \prec v$ dans r_1 et $v \prec u$ dans r_2. C'est-à-dire le nombre de paires dont l'ordre avant-après est inversé. Cela revient à utiliser une pénalité de $p = 0$ dans la définition 1.3.5. Notez qu'on utilise le terme "mesure" car on ne peut plus parler de distance, l'inégalité du triangle n'étant pas respectée (avec r_1 = [[1],[2]], r_2 = [[1,2]] et r_3 = [[2],[1]] on a $m(r_1,r_2) = 0$, $m(r_2,r_3) = 0$ mais $m(r_1,r_3) = 1$). Dans ce modèle, les égalités sont considérées comme un manque d'information dans lequel l'électeur ne peut pas différencier deux ou plusieurs candidats. Le problème d'agrégation consiste ici à trouver une permutation qui minimise la somme des mesures généralisées aux classements de l'ensemble d'entrée $\mathcal{R}$. L'opération de "bris" (*shattering*) consiste à créer un classement r_s à partir d'un classement r_1 et d'un autre classement r_2 tel que r_s reprend l'ordre des paires ordonnées avant-après dans r_1 et quand elles sont à égalité dans r_1, r_s reprend l'ordre de ces paires dans r_2 (noté $r_s \leftarrow r_2 * r_1$). Un algorithme de 2-approximation, **RepeatChoice**, qui généralise l'algorithme Pick-a-Perm [117] est présenté : un classement, qui au départ consiste en un seul grand panier contenant tous les éléments, est itérativement "brisé" (*shattered*) avec tous les classements d'entrée ($r' \leftarrow r_i * r$, $r_i \in R$). Un deuxième algorithme de $\frac{3}{2}$-approximation "**Ailon 3/2**" ou **LpKwikSort** se base sur la relaxation du modèle ILP. Dans cet algorithme, le modèle standard ILP relaxé est résolu. Les variables $x_{uv} \in [0,1]$, qui désignent l'ordre entre deux éléments dans la permutations, sont arrondies au sixième près à 0 ou à 1, celles qui sont $\frac{1}{6} < x_{uv} \leq \frac{5}{6}$ deviennent $\frac{3}{2}x_{uv} - \frac{1}{4}$. L'algorithme choisi un élément-pivot aléatoirement et

place tous les éléments soit à gauche soit à droite du pivot de façon probabiliste en fonction des variables arrondies. L'algorithme est appelé récursivement sur la partie gauche et la partie droite. Finalement, le travail discute de la complexité du problème et de d'autres métriques sur les classements.

Dans [19, 21], Betzler et al. présentent des algorithmes à paramètres fixes pour le problème du consensus de classements incomplets et avec égalités. Leur fonction de coût utilise $p = 1/2$ comme pénalité pour défaire une égalité. De plus, si un élément d'une paire n'est pas présent dans un classement alors cette paire est ignorée de la distance de Kendall-τ généralisée. Le problème de la médiane de classements avec égalité peut être résolu en $O((6d+2)! \times dlogd \times nm)$ et le problème de la médiane de classements incomplets peut être résolu en $O((1.48k)^k \times poly(n,m))$ (voir Section 2.6 pour les paramètres).

Une première heuristique, nommée **BioConsert**, spécialisée pour le problème du consensus de classements est donnée dans [40]. Dans ce travail, la distance de Kendall-τ généralisée est utilisée avec le paramètre $0 < p \leq 1$ comme pénalité de défaire des égalités. Le procédé d'unification est utilisé pour compléter les classements incomplets. Un panier est rajouté à la fin de chaque classement, contenant tous les éléments manquants. L'heuristique part d'un classement d'entrée comme classement de départ et effectue itérativement des opérations sur ce classement pour diminuer le score de Kemeny généralisé. Deux types d'opérations sont permises : 1) déplacer un élément d'un panier à un autre (*changeBucket*) et 2) créer un nouveau panier et déplacer un élément d'un panier existant dans ce panier (*addBucket*). Lorsqu'aucune opération diminuant le score est possible, l'heuristique garde le classement résultat comme potentiel candidat de consensus et recommence le processus avec un nouveau classement d'entrée. Lorsque chacun des classements d'entrée a été utilisé comme point de départ, l'heuristique retourne le classement candidat avec le score de Kemeny le plus bas. L'heuristique a été testé sur des petits classements aléatoires et sur des données biologiques. Les donnés biologiques proviennent de deux centres médicaux et sont des ensembles de quatre classements de gènes ayant trait à des maladies (voir Section 3.2). Des tests démontrent la supériorité de cette heuristique sur celles de [2] et de [59].

Dans [31], un premier travail de comparaison pour ce problème du consensus de classements utilisant la distance de Kendall-τ généralisée est présentée dans laquelle une pénalité $p = 1$ est donné pour briser une égalité et le score de Kemeny entre un classement et un ensemble de classement est la somme des distances généralisées. Il est montré que le problème n'est pas plus facile que celui de la médiane de permutations en utilisant un lemme d'un rapport interne [29]. Des adaptations de nombreuses heuristiques

du problème de la médiane de permutations au problème du consensus de classements sont étudiées. De plus, un important travail de classification et de comparaison entre ces heuristiques est fait. Il en ressort que très peu d'heuristiques existantes sont adaptables et celles qui le sont, sont difficilement applicables sur des instances de grandes tailles. Un premier modèle de programmation en nombres entiers (ILP) est donné pour ce problème (Voir Section 1.3.4) et une implémentation est faite utilisant CPLEX. L'article offre aussi un important ensemble de jeux de données (réelles ou synthétiques) sur lesquelles les tests sont fait. Finalement, l'article fait des recommandations sur le choix d'heuristiques ou d'algorithmes en fonction du temps alloué, de la qualité attendue du consensus, de la taille et de la similarité des données. Les méthodes qui en sortent gagnantes sont l'algorithme exact (ILP), l'heuristique BioConsert [40] et les généralisations des heuristiques KwikSort, BordaCount et Copeland.

2.11. Données et formats

Pour terminer ce chapitre de revue de littérature, cette section va brièvement faire le tour des données utilisées dans la littérature pour le problème de la médiane de permutations et du consensus de classements et de leurs formats.

2.11.1. Données réelles

Le premier type de données sont les données qui proviennent de la vie réelle ("real life data"). Elles sont tirées d'applications, de compétitions, de votes, de sondages, etc. Elles sont intéressantes pour évaluer des algorithmes destinés à des applications réelles et complémentent les données artificielles avec des distributions difficiles à générer artificiellement (par exemple des distributions multi-modales). Ce dernier point est intéressant quand on veut tester des idées ou des conjectures.

Une source majeure et importante de données réelles est **Preflib** [91] qui est une librairie de données de préférences disponible sur le web à l'adresse http://www.preflib.org/. PrefLib contient la majorité des données trouvées dans les articles. Elle contient aussi des données d'association (*matching*) et des graphes de tournois. Les données de PrefLib sont utilisées, entres autres, dans les articles [17, 18, 49].

Lorsque les données réelles ne sont pas prise sur PrefLib, elles peuvent venir de sources très diverses dont une grande partie sont présentées dans l'énumération suivante :

1. **Ski** : Les données provenant de compétitions de ski sont utilisées dans [6] et [18]. Les auteurs de [6] utilisent la saison 2008-2009 de Giant Slalom chez les femmes

contenant 16 courses. Ces données peuvent être retrouvée sur : http://www.ski-db.com/. Les auteurs de [18] utilisent, quand à eux, les classements finaux de Ski Jump et Ski Cross des coupes mondiales de 2005/2006 à 2008/2009. Ces données peuvent être retrouvée sur : http://www.sportschau.de/sp/wintersport/.

2. **Formule 1** : Les données provenant de courses automobiles de la Formule 1 sont utilisées dans [18]. Elles contiennent les saisons s'étendant de 1970 à 2008. Chaque saison contient plusieurs courses qui peuvent être interprétées comme des classements. Les auteurs utilisent la projection (Section 1.3.5) pour garder seulement les pilotes de F1 qui ont participé à toutes les courses.

3. **Web Search** : Comme l'une des applications de la méthode de Kemeny est basée sur la construction d'un métamoteur de recherche, plusieurs auteurs ont pris des données provenant de requêtes envoyées à divers moteurs de recherche. Dans les articles [6, 18, 56, 117], 37 requêtes[3] sont envoyée à 4 moteurs de recherche commerciaux[4]. La procédure d'unification (Section 1.3.5) est généralement utilisé dans ce cas en ajoutant à la fin des classements les résultats absents mais présents dans les autres moteurs.

4. **Web impact** : Dans [18], les auteurs obtiennent aussi des classements en demandant à des moteurs de recherche de retourner une liste des capitales du monde, des nations et des personnes les plus riches.

5. **Web communities** : Dans [117], les auteurs utilisent des données provenant de [41] et consistant en des ensembles de 9 classements de 100 documents issus de l'internet.

6. **Irish election** : Dans [48], les auteurs utilisent les résultats d'un sondage réalisé avant les élections irlandaises de 1997 qui a donné 1083 classements, dont certains incomplets, sur 5 candidats.

7. **European Values Study** : Dans [48], les auteurs utilisent les résultats d'un second sondage : une étude réalisée en 1999 sur certains enjeux de société en Europe et qui a donné 3584 classements sur 4 enjeux.

3. "Affirmative action, Alcoholism, Amusement parks, Architecture, Bicycling, Blues, Cheese, Citrus groves, Classical guitar, Computer vision, Cruises, Death valley, Field hockey, Gardening, Graphic design, Gulf war, HIV, Java, Lipari, Lyme disease, Mutual funds, National parks, Parallel architecture, Penelope Fitzgerald, Recycling cans, Rock climbing, San Francisco, Shakespeare, Stamp collecting, Sushi, Table tennis, Telecommuting, Thailand tourism, Vintage cars, Volcano, Zen buddhism, Zener"

4. "Google, Lycos, MSN Live Search, Yahoo"

8. **Sushis** : Un sondage de 2003 sur les préférences alimentaires de 5000 personnes classe 10 aliments utilisés dans les sushis. Il peut être retrouvé sur http://www.kamishima.net/sushi/ et a été utilisé dans [49].
9. **American Psychological Association** : En 1980, l'American Psychological Association procédait à une élection pour élire un nouveau président de l'association. Cela regroupait plus de 15000 électeurs et 5 candidats [8, 48].
10. **U Illinois Sports** : Ici, l'ensemble de données est celui d'un sondage réalisé à l'Université d'Illinois dans lequel 130 étudiant ont classé les 7 sports proposés selon leur préférences [8, 48].
11. **50 American states** : Un dernier ensemble de données est celui du classement des 50 états américains en fonction de facteurs socio-économiques, de santé et de criminalité [101]. Dans cet ensemble, les 50 états sont ordonnés dans 104 classements. On trouve son utilisation dans [8, 49].

Ils semble qu'il soit plus facile de calculer la médiane de données réelles, étant donné la plus grande ressemblance entre les permutations. La grande majorité des auteurs discutent de la performance de leurs algorithmes, heuristiques ou autres méthodes avec des données réelles. Une moindre partie ajoute des données artificielles à leurs expériences.

2.11.2. Données artificielles

Le second type de données sont les données artificielles ou synthétiques ("artificial data" et "synthetic data"). Ces données sont générées selon différents modèles probabilistes. Elles sont particulièrement intéressantes pour tester des algorithmes de façon massive, les paramétrer automatiquement et évaluer leur performance selon les paramètres des modèles.

Le mélange **Fisher-Yates** [64] est une méthode très simple qui permet de générer aléatoirement une permutation d'une taille donnée. Dans la première étape, une liste est créée qui contient les éléments 1 à n ordonnés en ordre croissant. En deuxième étape, les positions sont parcourues itérativement de la dernière à la première. À chaque position i parcourue, on pige aléatoirement et uniformément une seconde position j telle que $1 \leq j \leq i$ et on échange ces deux éléments dans les positions i et j. Il est prouvé que cette génération aléatoire est uniforme dans l'espace des permutations S_n. Le modèle **aléatoire uniforme** consiste simplement à générer un ensemble de m permutations utilisant le mélange de Fisher-Yates. On considère que les ensembles obtenus par ce modèle sont parmi les plus

difficiles à résoudre en raison de leur absence générale de similarité. Ce modèle a été utilisé dans [6, 48, 96, 97] et [98].

Un ensemble de données tiré du modèle aléatoire uniforme est disponible sur GitHub [5]. Ces données contiennent le score de Kemeny optimal et une permutation médiane pour chaque ensemble. Elles peuvent alors être utilisées pour tester des heuristiques ou algorithmes. Les ensembles vont de 3 à 15 permutations de taille allant entre 10 et 110 éléments.

Le modèle **Mallows-ϕ** de Mallows [89] définit une distribution dans l'espace des permutations selon une permutation centrale π et un paramètre de dispersion $\lambda \geq 0$ dans laquelle la probabilité d'observer une permutation σ est donnée par :

$$P_{\pi,\lambda}(\sigma) = \frac{exp(-\lambda \times d_{KT}(\pi,\sigma))}{C} \quad \text{où} \quad C = \sum_{\mu \in S_n} exp(-\lambda \times d_{KT}(\pi,\mu)).$$

Plus $\lambda \in \mathbb{R}_+$ est grand, plus les permutations de la distribution sont rapprochées de la permutation centrale. Si $\lambda = 0$, alors les permutations sont complètement aléatoires et cela revient au modèle aléatoire uniforme précédent. La constante C sert simplement à uniformiser pour que la somme de toutes les probabilités donne 1. Ce modèle a été utilisé dans [6, 8] et [18].

Il est intéressant de noter que la permutation médiane est l'estimateur de maximum de vraisemblance de la permutation centrale π dans ce modèle [92] et qu'on peut estimer le paramètre de dipersion λ si on connaît la permutation médiane en résolvant l'équation suivante [33, 65] :

$$\bar{d}_{\mathcal{A}} = \frac{ne^{-\lambda}}{1-e^{-\lambda}} - \sum_{j=1}^{n} \frac{je^{-j\lambda}}{1-e^{-j\lambda}},$$

où $\bar{d}_{\mathcal{A}}$ est la moyenne des distances entre les permutations de $\mathcal{A}$ et la permutation médiane et où n est la taille des permutations. Cette équation n'a pas de forme close mais peut être résolue par une méthode itérative.

Le modèle **Plackett-Luce** [87, 108] définit une distribution dans l'espace des permutations selon un vecteur $S = \{s_1,...,s_n\} \in \mathbb{R}_+^n$ qui attribue des poids aux n éléments, représentant l'importance de l'élément. Plus un élément est important, plus il a de chances de se trouver au début de la permutation. La probabilité d'avoir la permutation σ en fonction du vecteur S est :

5. https://github.com/robinmilosz/KemenyConsensusExamples

$$P_S(\sigma) = \prod_{i=1}^{n} \frac{s_{\sigma_i}}{Z_i} \quad \text{où} \quad Z_i = \sum_{i \leq j}^{n} s_{\sigma_j}.$$

Ce modèle a été utilisé dans [6] et [18].

2.11.3. Formats des données

Plusieurs formats de données sont retrouvés dans la littérature du problème de la médiane. Certains sont conçus pour accepter les classements avec égalités et certains peuvent contenir des identifiants non-numériques.

L'exemple d'introduction présenté à la Figure 1.1 sera utilisé pour présenter chaque format. Les cinq activités suivantes sont ordonnées par trois personnes : 1) faire un projet de peinture, 2) jouer au ballon, 3) aller au cinéma, 4) visiter le centre des sciences et 5) aller au parc nature. Les mots surlignés représentent les identifiants non-numériques et les identifiants numériques précèdent chaque activité.

Le premier format **l'ensemble de permutations** est un format simple qui provient de la notation mathématique d'un ensemble de permutations :

```
{[4,5,1,2,3],[1,5,3,4,2],[5,2,3,4,1]}
```

Un version simplifiée et plus compacte omet les virgules et les crochets dans les permutations :

```
{45123,15342,52341}
```

Ce format est utilisé dans [26] et [96].

Le seconde format, **l'ensemble de classements**, généralise le premier format aux classements. Celui-ci peut contenir des éléments à égalité et s'écrit comme un ensemble de liste ordonnées d'ensembles d'éléments :

```
{[{4},{5},{1},{2},{3}],[{1},{5},{3},{4},{2}],[{5},{2},{3},{4},{1}]}
```

On retrouve ce format dans le projet BioConsert [40] http://bioguide-project.net/bioconsert/ et dans [30] qui introduit conqur-bio.lri.fr.

Le troisième format est celui de **la liste de classements specifiés**. Dans ce format, chaque classement a un identifiant propre et les identifiants non-numériques sont acceptés :

```
R1 := [[Sciences],[Nature],[Peinture],[Ballon],[Cinéma]]
R2 := [[Peinture],[Nature],[Cinéma],[Sciences],[Ballon]]
R3 := [[Nature],[Ballon],[Cinéma],[Sciences],[Peinture]]
```

La version numérique est :

```
R1 := [[4],[5],[1],[2],[3]]
R2 := [[1],[5],[3],[4],[2]]
R3 := [[5],[2],[3],[4],[1]]
```

On retrouve ce format sur la plateforme CoRankCo disponible à l'adresse https://corankco.lri.fr/.
Le quatrième format est **la liste de préférences**. Dans ce format, chaque ligne représente un vote ou une permutation et les éléments sont séparés par des signes > représentant la préférence. Les identifiants non-numériques sont aussi acceptés :

```
Sciences > Nature > Peinture > Ballon > Cinéma
Peinture > Nature > Cinéma > Sciences > Ballon
Nature > Ballon > Cinéma > Sciences > Peinture
```

La version numérique est :

```
4 > 5 > 1 > 2 > 3
1 > 5 > 3 > 4 > 2
5 > 2 > 3 > 4 > 1
```

Ce format est retrouvé dans [**18**].
Le dernier format est celui des **élections PrefLib** [**91**] http://www.preflib.org/ (voir Section 2.11.1). Il y a quatre variantes de type de données qui nous intéressent ici :

— "soc" ordres stricts, complets

— "soi" ordres stricts, incomplets
— "toc" ordres avec égalités, complets
— "toi" ordres avec égalités, incomplets

Les ensembles de permutations sont de type "soc".
Le format de PrefLib est un peu plus élaboré que les autres formats. La première ligne d'un fichier indique le nombre n de candidats (éléments). Les candidats sont ensuite énumérés avec des identifiants numériques associés. La ligne suivante indique le nombre total d'électeurs m, la somme des votes et le nombre de votes uniques. Dans notre cas, le nombre total d'électeurs et la somme des votes sont identiques, alors que pour d'autres types de données ces deux mesures peuvent être différentes (par exemple, quand les électeurs expriment plusieurs votes). Chacun des différent votes uniques est ensuite énuméré, précédé par le nombre de fois où ce vote apparaît dans l'élection. On peut considérer ce nombre comme le poids associé au vote.

```
nombre de candidats
ID, nom du 1er candidat
ID, nom du 2e candidat
...
nombre total d'électeurs, somme des votes, nombre de votes uniques
nombre de votes, 1er vote
nombre de votes, 2e vote
...
```

Pour notre exemple d'activités, cela donne :

```
5
1, Peinture
2, Ballon
3, Cinéma
4, Sciences
5, Nature
3,3,3
1,4,5,1,2,3
```

```
1,1,5,3,4,2
1,5,2,3,4,1
```

Pour les votes avec égalités (toc, toi), les éléments à égalité vont se retrouver dans le même ensemble dénoté par des accolades (i.e. 1,4,{2,3,5}). Pour les votes incomplets (soi, toi), les éléments absents ne sont simplement pas présentés dans ces votes.

Dans cette section ont été présentées plusieurs sources de données, quelles soient de provenance réelle ou artificielle, et différents formats utilisés pour représenter ces données. Le chapitre présent a fait un tour de travaux qui ont été réalisés mais seulement au niveau de la compréhension et de la résolution du problème. Le prochain chapitre va, quant à lui, présenter diverses applications du problème de la médiane de permutations et du consensus de classements qui originent de plusieurs domaines en sciences.

Algorithme	Articles	Schalekamp et van Zuylen 2009 [117]	Ali et Meila 2012 [6]
Pick-a-perm	[117]	x	x
BestOfA (best-of-k)	[117]	x	x
Random permutation	[117]	x	
Borda	[27]	x	x
Copeland	[45]	x	x
CopelandWins	[42]		
Schulze	[119]		
RankedPairs	[130]		
Plurality	[112]		
Raynaud	[112]		
Minimax	[112]		
MC4	[56]	x	x
MC4Approx	[117]	x	x
Footrule aggregation	[54]	x	x
MEDRank	[61]		
CSS	[39]		x
Greedy graph heuristic	[51]		
KwikSort	[3]	x	x
DetQuickSort	[117]	x	x
LogQuickSort	[117]	x	x
MergeSort	[117]	x	x
InsertionSort	[117]	x	x
Chanas	[35]	x	x
ChanasBoth	[41]		
Beam search	[92]		x
LPRelaxation (borne inf.)	[6, 44, 117]	x	x
LpKwikSort (Ailon-3/2)	[2]		
FindMedian	[25]		
VNS	[15]		
QUICK	[8]		
FAST	[8]		
Subiterative Convergence	[12]		
Greedy Algorithm	[12]		
FUR	[12]		
SIgFUR	[12]		
GA	[4]		
DECoR	[49]		
Simulated Annealing	[97]		

TABLEAU 2. III. Récapitulatif des heuristiques présentées dans la Section 2.4. Les "x" indiquent les heuristiques comparées dans les travaux de Schalekamp et van Zuylen 2009 [117] et de Ali et Meila 2012 [6].

Algorithme	Articles	Schalekamp et van Zuylen 2009 [117]	Ali et Meila 2012 [6]
LocalSearch	[6, 117]	x	x
IS	[6, 117]	x	x
MS	[6, 117]	x	x
KS	[6, 117]	x	x
2-opt (Local Kemenization)	[56]		
3-opt et k-opt	[95]		

TABLEAU 2. IV. Récapitulatif des heuristiques de raffinement présentées dans la Section 2.4. Les "x" indiquent les heuristiques comparées dans les travaux de Schalekamp et van Zuylen 2009 [117] et de Ali et Meila 2012 [6].

Algorithme	Articles	Schalekamp et van Zuylen 2009 [117]	Ali et Meila 2012 [6]
branch-and-bound	[92]		x
branch-and-bound	[44]		
branch-and-bound	[48]		
branch-and-bound	[97]		
ILP avec CPLEX	[6, 44, 117]	x	x

TABLEAU 2. V. Récapitulatif des algorithmes exactes présentées dans la Section 2.3. Les "x" indiquent les algorithmes comparés dans les travaux de Schalekamp et van Zuylen 2009 [117] et de Ali et Meila 2012 [6].

Chapitre 3

APPLICATIONS

Nous présentons ici plusieurs applications du problème de la médiane de permutations et du problème du consensus de classements. Dans ce contexte, le nom de méthode de Kemeny est majoritairement utilisé dans les travaux.

Notez que plusieurs variantes de la méthode de Kemeny-Young sont utilisées par les auteurs de ces applications. Par exemple, quand ce sont seulement les premiers candidats qui sont d'intérêt, les auteurs peuvent utiliser les "top-k lists" qui ne considèrent que les premiers k candidats de chaque permutation, alors que quand les candidats sont nombreux, les auteurs vont souvent utiliser des heuristiques (Section 2.4) pour trouver rapidement une approximation de la médiane. On considère tout de même que ces applications sont d'intérêt pour l'étude de la médiane de permutations.

Il est intéressant de noter que malgré le fait que les méthodes d'agrégations qui associent des points aux candidats en fonction de leurs positions (i.e. la méthode de Borda, Section 2.4.2) sont plus simples à comprendre et à implémenter, la méthode de Kemeny-Young reste quand même bien utilisée dans plusieurs domaines. Plusieurs auteurs d'articles montrent que cette dernière méthode performe généralement bien et souvent mieux que d'autres méthodes d'agrégations dans leurs contextes. Finalement, les propriétés mathématiques solides de la médiane de permutations (voir 2.1) et le modèle de Mallows [**89**] lié au problème (voir Section 2.11.2) sont souvent appréciées par les auteurs.

3.1. Système électoral

Un système électoral est une méthode qui permet de décider d'un choix à partir de préférences individuelles d'un ensemble d'électeurs. En politique, la méthode de Kemeny-Young est un système électoral [**140**] qui permet d'ordonner une liste de candidats (par opposition à un système à un seul gagnant) et qui possède plusieurs qualités désirables dont le critère de la majorité et le critère de Condorcet (voir 2.1).

Malgré le fait que cette méthode n'est utilisée dans aucun système électoral officiel, il est toutefois pertinent de s'y intéresser. Récemment, en 2011, le Royaume-Uni a eu un référendum [1] [2] sur son mode de scrutin pour élire les députés de la Chambre des communes. La méthode utilisée alors était le vote unique majoritaire à un tour dans lequel était déclaré gagnant le candidat ayant obtenu le plus de votes. La méthode alternative proposée au référendum s'énonçait ainsi : Chaque électeur ordonne des candidats (tous les candidats ou seulement un sous-ensemble) par rapport à ses préférences. Seulement le premier candidat de chaque bulletin de vote est considéré. Si un candidat obtient plus de la moitié des votes alors il est élu, sinon, le candidat ayant le moins de votes est disqualifié et les votes obtenus par ce candidat sont redistribués aux autres candidats en fonction du prochain candidat inscrit sur le bulletin de vote qui est encore présent dans l'élection. Le processus se répète jusqu'à ce qu'un candidat accumule plus de la moitié des votes. Finalement, le nouveau mode de scrutin n'a pas été accepté, avec plus du deux tiers des électeurs contre ce système alternatif et un taux de participation de 42.2%. Le sujet du choix du système électoral est donc d'actualité. De plus, dans le cas du Royaume-Uni en 2011, le système alternatif proposé était un système préférentiel.

3.2. Bio-informatique et sciences biomédicales

En bio-informatique, la médiane de permutations peut servir à classer les protéines/gènes liées à une maladie et à la construction de cartes génétiques.

Dans [**121**] (2001), les auteurs s'intéressent à l'expression des gènes (quantifier les ARN messagers dans un tissu donné à un moment donné). Plusieurs méthodes peuvent être utilisées pour mesurer cette expression (puce à ARN *microarray*, SAGE, BodyMap, MPSS). Les auteurs transforment les résultats de ces méthodes, qui quantifient les occurrences d'ARNm, en classements, et utilisent la médiane de permutation pour agréger ces classements.

L'expression des gènes a aussi été étudiée dans [**52**] (2006). Les données provenant de puces à ARN peuvent être représentées par des classements qui ordonnent l'importance de l'implication d'un gène dans une maladie. À travers plusieurs études de transcriptomique sur la même maladie, les conditions peuvent changer impliquant une variation importante

1. https://www.electoralcommission.org.uk/___data/assets/pdf_file/0019/141328/Final-PVS-report.pdf
2. https://web.archive.org/web/20110410141102/http://www.aboutmyvote.co.uk:80/PDF/England-accessible.pdf

des niveaux d'expression des gènes. Pour agréger les résultats de plusieurs études, l'utilisation de la médiane de permutations est utilisée car elle ne tient compte que des ordres relatifs entre les gènes. Dans ce travail, des données ont été prises de cinq études sur le cancer de la prostate. Les auteurs ont été surpris par le peu de chevauchement entre les top-25 des classements provenant de chaque étude, i.e. les diverses méthodes n'identifient pas les mêmes cibles comme les plus importantes. L'agrégation a permis de mettre de l'avant plusieurs gènes déjà identifiés pour leur implication dans la maladie et d'autres gènes reliés à différents mécanismes cellulaires.

Une version modifiée du problème de la médiane de permutations est utilisée pour la construction de cartes génétiques dans [**73**] (2008). Dans ce contexte, des marqueurs génétiques de chaque population d'une espèce représentent des ordres partiels (certains marqueurs peuvent être absents dans une population) et les chercheurs s'intéressent à trouver un ordre total des marqueurs génétiques qui représente le mieux l'espèce en entier. Dans ce travail, les auteurs introduisent des poids liés au niveau de confiance des paires ordonnées, décrivent une heuristique basée sur le Minimum Feedback Arc Set problem (voir début du Chapitre 2) et testent leur approche sur des données provenant du maïs. Notez que le résultat de leur heuristique est un ordre partiel.

Dans [**40**] (2011), les auteurs utilisent l'heuristique BioConsert (voir Section 2.10) sur des données biologiques. Ces donnés proviennent de deux centres médicaux et sont des ensembles de quatre classements de gènes ayant trait à chacune des maladies suivantes : le cancer du sein, le cancer de la prostate, le cancer de la vessie, le neuroblastome, le rétinoblastome, le TDAH et le syndrome du QT long. Les quatre classements proviennent de quatre méthodes de prédiction de gènes liés à une maladie. Ces données sont disponibles sur http://bioguide-project.net/bioconsert/. Les classements consensus donnés par BioConsert sont bien pertinents : les gènes qui se retrouvent au début des consensus ont un nombre plus élevé de publications qui les associent avec la maladie. Finalement, le consensus permet de corriger certaines anomalies retrouvées dans les résultats de certaines méthodes de prédictions : des gènes qui sont classés dans les 20 premiers et qui ne semblent pas liées à la maladie. Cette agrégation de résultats de méthodes de prédictions semble être très pertinente dans un contexte de recherche en laboratoire.

Dans [**10**] (2011), les auteurs utilisent la médiane de permutations pour la prédiction de conformations d'amarrage de protéines. L'objectif est de prédire avec des outils informatiques des conformations acceptables, c'est-à-dire des conformations proches des

observations réelles (par cristallographie aux rayons X par exemple). Il est facile de générer rapidement beaucoup de conformations possibles, par contre, la seconde étape, qui consiste à filtrer les faux positifs avec des algorithmes plus précis, est plus difficile à réaliser. Les auteurs utilisent Hex [**111**] qui peux générer une grande quantité de conformations et considèrent seulement les 50 meilleures conformations données par ce logiciel. L'objectif est alors de reclasser ces 50 conformations pour que celles qui sont plus pertinentes se retrouvent mieux classées. Les auteurs utilisent une approche par diagrammes de Voronoi pour modéliser les conformations et calculent les attributs et les propriétés chimiques des conformation. Après normalisation, ces propriétés sont données à plusieurs algorithmes d'apprentissage qui prédisent avec des probabilités si ces conformations d'amarrage sont acceptables. Des permutations sont obtenues en classant les conformations en fonction de leurs probabilités d'être acceptables pour chaque méthode. L'heuristique FindMedian [**26**] est ensuite utilisée pour approximer la méthode de Kemeny sur ces permutations de taille $n = 50$. Les résultats montrent que le consensus trouvé par leur approximation donne des meilleurs classements des conformations que les méthodes individuelles. Entre autres, les conformations acceptables sont classées plus haut en moyenne que les autres méthodes. De plus, les auteurs montrent que les conformations bien classées par leur méthode d'agrégation sont plus pertinentes biologiquement.

Les microARNs sont des petites séquences d'ARN non-codant qui peuvent jouer des rôles de régulation sur l'expression des gènes. Dans [**120**] (2013), les auteurs s'intéressent à la détection des gènes ciblés par des microARN. Pour agréger les résultats de plusieurs algorithmes de prédiction des cibles, les auteurs utilisent une méthode dérivée de la méthode de Kemeny-Young dans laquelle ils introduisent un facteur lié à la dispersion des distances entre le consensus et les permutations d'entrée (utilisant l'entropie de Shannon [**122**]). Leur objectif est de favoriser un consensus qui ne minimise pas seulement les désaccords mais qui est équitable envers chaque entrée. Une heuristique de recuit simulé est décrite pour résoudre ce problème : l'échange de positions d'éléments adjacents est utilisé pour générer des nouvelles permutations voisines et la température descend par paliers. Les auteurs comparent leur méthode à la méthode de Kemeny-Young et d'autres méthodes d'agrégation en la testant sur des données de microARNs et affirment que leur méthode est plus performante.

Un outil web de bio-informatique nommé **Conqur-bio** (conqur-bio.lri.fr) a été développé dans [**30**] (2014) et est un bel exemple d'application du consensus de classements. Dans un contexte de croissance exponentielle des bases de données de bio-informatique,

cet outil utilise le consensus de classement pour aider les utilisateurs à retirer les informations les plus pertinentes de ces bases de données. Plus précisément, un utilisateur serait intéressé à entrer des mots clés dans le moteur de recherche de la base Entrez Gene [**88, 116**] de NCBI (https://www.ncbi.nlm.nih.gov/gene/). Il peut alors faire cette requête avec les mêmes mots clés sur Conqur-bio, qui génère des synonymes des mots clés, fait plusieurs requêtes à Entrez avec différentes combinaisons des synonymes, agrège les résultats et retourne le consensus. Cela permet entre autres de recueillir les résultats qui ne seraient présents qu'avec un autre synonyme de langue (*tumor* vs *tumour*), une autre abréviation ou terme complet(*ADHA* vs *Attention Deficit Hyperactivity Disorders*), une reformulation équivalente (*cervix cancer* vs *cervical cancer*) ou une reformulation avec des termes ou sujets plus précis (*colorectal cancer* vs *hereditary nonpolyposis colon cancer* ou *lynch syndrome*). Dans tous ces cas énumérés, il y a eu des gains de pertinence dans les résultats retournés ; certains gènes sont apparus dans les résultats alors qu'ils n'étaient pas présents dans le résultat de la requête originale. Dans la méthode d'agrégation, l'outil utilise l'unification pour avoir des classements complets en ajoutant un panier à la fin de chaque classement. Ce dernier panier a un statut spécial dans la mesure utilisée qui ne tient pas compte de l'ordre entre ses éléments. Une égalité entre deux éléments dans un panier normal correspond à un moteur de recherche qui a classé ces deux éléments à égalité alors qu'un égalité dans le panier d'unification indique que le moteur de recherche n'a pas d'information différenciant ces deux éléments. La mesure utilisée ressemble fortement à la distance de Kendall-τ généralisée avec $p = 0.5$ avec la seule différence que si deux éléments sont à égalité dans le panier d'unification dans un classement alors qu'ils ne sont pas à égalité dans l'autre classement alors il n'y a pas de coût p ajouté à la mesure pour cette paire. Conqur-bio utilise les algorithmes de BordaCount [**27**], MEDRank [**61**] et Ailon (Pick-a-Perm généralisé) [**2**] pour donner rapidement un résultat d'une qualité acceptable. L'outil a été testé sur plusieurs données de provenance biologique.

Une méthode d'agrégation de données génomiques nommée **HyDRA** est introduite dans [**81**] (2015). HyDRA est une méthode hybride qui utilise l'agrégation avec score et une version modifiée (avec poids) de la méthode de Kemeny pour agréger des listes de gènes en fonction de leur association avec une maladie dans la littérature. L'objectif de la méthode HyDRA est de découvrir des nouveaux gènes susceptibles d'être associés à une maladie ciblée mais qui n'ont pas encore été investigués dans le cadre de recherche

dans cette maladie. Les auteurs comparent HyDRA avec les outils Endeavour [3] [**131**] et ToppGene [4] [**37**].

Pour aider la recherche en science biomédicale, un programme de recommandation d'articles a été fait dans [**107**] (2017). La plateforme **Meta** (https://meta.com/) puise dans les articles et les bases de données Crossref [5] et Pubmed [6] pour proposer des articles à lire à des chercheurs dans différents domaines des sciences biomédicales. Meta utilise plusieurs algorithmes indépendants qui donnent des recommandations en fonctions de graphes de citations ou d'auteurs, de listes de mots-clés, de similarité de références, de sémantique et autres données. Les recommandations individuelles de ces algorithmes sont alors agrégées pour donner une liste de recommandations finale. Les auteurs montrent que chaque algorithme indépendant utilisé ne couvre pas tous les articles pertinents à suggérer.

3.3. Sciences médicales

En sciences médicales, la méthode de Kemeny-Young peut servir à améliorer certaines pratiques médicales ou administratives. Dans [**109**] (2009), cette méthode a été utilisée avec deux autres méthodes pour ordonner les préférences des cliniciens d'hôpitaux au sujet des formats et des contenus des rapports de radiologie. Les mêmes auteurs continuent dans [**68**] (2010) en analysant les préférences des médecins généralistes sur les rapports de radiologie. L'objectif de ces deux études était de construire un rapport modèle de radiologie pour les médecins que les radiologues pourraient suivre.

La méthode de Kemeny-Young a encore été utilisée dans [**132**] (2014) pour agréger les préférences d'oncologues et de radiologistes sur des formats de rapports de radiologie.

3.4. Sciences sociales

La méthode de Kemeny-Young peut aussi servir à construire un consensus d'estimations de valeurs (dimensions, temps) d'objets ou d'événements, données par un groupe d'individus. Les auteurs de [**94**] (2009) ont étudié plusieurs méthodes de consensus dont la méthode de Kemeny-Young. L'étude utilise une version plus simple de la méthode de

3. https://endeavour.esat.kuleuven.be/
4. https://toppgene.cchmc.org/prioritization.jsp
5. https://www.crossref.org/
6. https://www.ncbi.nlm.nih.gov/pubmed/

Kemeny-Young en choisissant la permutation consensus à partir de l'ensemble des permutations d'entrée (ce qui revient à l'heuristique BestOfA présentée à la Section 2.4). 78 participants ont répondu à une vingtaine de questions portant sur l'ordonnancement de sujets liés aux populations, à la géographie, aux dates, à l'histoire, à la dureté des matériaux et aux lois. Pour chaque question, les participants devaient ordonner des concepts ou des sujets selon une caractéristique telle que la taille, l'ancienneté, etc.. Les consensus ont ensuite été calculé à partir de plusieurs méthodes et ils ont été comparés au "vrai" classement, i.e. l'ordonnancement réel des objets ou événements. La méthode de Kemeny-Young a fini en première place. Les critères utilisés étaient 1) la distance de Kendall-τ entre le consensus et le vrai classement et 2) le pourcentage d'électeurs qui ont des classements "moins bons" que le consensus.

3.5. Sciences environnementales et agricoles

Dans [**112**] (2016), la nuisance sonore provenant des routes est analysée et différentes variables qui influencent cette nuisance sont étudiées. Pour déterminer les variables les plus importantes dans cette problématique, les auteurs ont proposé 9 variables candidates et ont recueillit l'avis de 19 experts du domaine de leur région (Andalousie, Espagne). Plusieurs méthodes d'agrégations des permutations ont été utilisées pour obtenir un consensus : Borda, Copeland, Kemeny-Young, Schulze, Raynaud, Simpson et Plurality (Section 2.4). Alors que les avis des experts avaient tendance à différer, les résultats des méthodes d'agrégations étaient plutôt similaires.

Dans [**70**] (1994), la méthode de Kemeny est utilisée pour trouver un ordonnancement de choix de variétés de semences. Plusieurs méthodes classent des variétés de semences en fonction de critères tels que la productivité moyenne et sa variabilité, la résistance aux maladies, à la sécheresse et aux inondations, le coût et les besoins en nutriments. Les auteurs étudient le colza avec 14 variétés et 60 classements. Un algorithme de branch-and-bound est discuté. Cet algorithme combine une structure de données qui empêche de recalculer certaines sous-arborescences. Les auteurs sont particulièrement intéressés par les objets classés parmi les meilleurs des médianes. Ils concluent que la méthode de Kemeny-Young est très utile quand on veut comparer plusieurs objets sur la base de plusieurs critères qui ne couvrent potentiellement pas tous les objets.

3.6. Sciences informatiques

En informatique, la médiane de permutations peut servir à la métarecherche sur le web par agrégation de résultats de plusieurs moteurs de recherche [**56**] (2001). Dans ce dernier travail, un site web bien classé par un seul moteur de recherche est considéré comme du "spam".

La méthode de Kemeny-Young est utilisée dans [**126**] (2014) pour trouver un consensus de plusieurs méthodes d'analyse de similarité entre réseaux. Une méthode d'analyse de similarité entre réseaux prend en entrée un réseau cible et un ensemble de réseaux et ordonne les réseaux de l'ensemble en fonction de la similarité avec le réseau cible. Dans ce travail, 20 méthodes d'analyse sont considérées. Un consensus est calculé à partir de tous les classements donnés par ces méthodes. Avec l'hypothèse que chaque classement est une observation bruitée du "vrai" classement, les auteurs montrent que certaines méthodes donnent souvent des classements plus rapprochés du consensus et ces méthodes sont alors jugées meilleures.

Dans [**127**] (2014), la méthode de Kemeny-Young est aussi utilisée pour faire l'agrégation de résultats de plusieurs algorithmes qui comparent les *workflows* (flux de travail) scientifiques. Les *workflows* sont exprimés comme des graphes orientés acycliques et ils représentent des suites d'opérations et de données. Plusieurs algorithmes existent pour comparer les *workflows* et se basent sur des éléments structurels (topologie, modules, chemins, distance d'édition de graphe) ou les annotations (étiquettes). Ils sont utilisés, entre autres, pour regrouper les *workflows* par similarité et pour détecter des *workflows* équivalents. Dans ce travail, on a demandé à 15 experts du domaine d'ordonner un échantillon de *workflows* en fonction de la similarité avec un workflow requête. Les auteurs utilisent l'heuristique BioConsert (voir Section 2.10) pour trouver un consensus des classements donnés par des experts et évaluent les résultats des algorithmes contre ce consensus.

Un article intéressant [**129**] (2011) montre l'utilité de l'agrégation de Kemeny dans les réseaux sociaux pour tenter de prédire les personnes les plus influentes. Dans ce travail, effectué sur les données de plus de 40 millions d'utilisateurs de Twitter, plusieurs mesures sont évaluées pour prédire les prochains influenceurs (les utilisateurs qui sont le plus "retweetés") : le nombre d'abonnés, le nombre d'amis, le nombre de messages "retweeté", le nombre de mentions, etc. puis des combinaisons de ces mesures et l'algorithme PageRank [**105**]. Chacune des 13 mesures ordonne les utilisateurs en ordre décroissant dans une liste (permutation) et une agrégation de ces listes est recherchée. Contrairement, au système de vote où l'égalité entre les votes est importante, dans ce contexte, les auteurs cherchent à

pondérer les votes (les mesures) pour avoir la meilleure prédiction. Les auteurs considèrent aussi seulement les meilleurs candidats de chaque liste ordonnée par un critère et utilisent une version modifiée de QuickSort pour produire une approximation rapide. Leur méthode, "Supervised Kemeny Ranking", (SKR) qui utilise ces stratégies, est paramétrée par les niveaux de performance individuels de chaque mesure. Les auteurs comparent SKR à plusieurs autres méthodes, dont Borda et des variantes supervisées, et à la mesure du nombre de messages retweets (la meilleure mesure) puis montrent que leur méthode est la plus performante pour prédire les prochains grands influenceurs de réseaux sociaux. Ils montrent que les méthodes basées sur Kemeny sont plus performantes que celles basées sur un score (Borda).

Un autre article [**110**] (2012), très similaire au dernier mentionné, a travaillé sur les données de DBLP (base de données des auteurs et ouvrages en science informatique). Une méthode assez similaire "Supervised local Kemeny Aggregation" est décrite. Cette dernière utilise BubbleSort à la place de QuickSort.

L'article [**1**] (2012) utilise aussi le "Supervised Kemeny Ranking" dans le cadre des systèmes experts pour répondre à des questions de connaissance générale ou médicale. Dans ce contexte, un processus de traitement naturel des langues (NLP) analyse une question donnée et cherche des candidats de réponses utilisant des bases de données de connaissance. Cinq algorithmes ordonnent les candidats en fonction de leur pertinence évaluée. Les données proviennent du jeu de génie en herbe *Jeopardy* et de *Doctor's Dilemma*. L'article montre que les méthodes de consensus sont en général plus efficaces qu'un seul algorithme de classement. Les auteurs recommandent d'ailleurs d'utiliser une méthode d'agrégation à la place du meilleur algorithme d'ordonnancement. Leur méthode, basée sur la méthode de Kemeny, effectue un premier ordonnancement des candidats et demande ensuite aux cinq algorithmes d'ordonner les top-k candidats puis fait l'agrégation. D'après leurs expérimentations, leur méthode donne les meilleurs résultats.

3.7. Sports

Malgré le fait que la méthode de Kemeny-Young n'est pas utilisé en pratique dans les sports, plusieurs scientifiques utilisent cette méthode sur des données issues du monde sportif et la comparent avec les méthodes officielles utilisées dans les compétitions par les juges pour désigner les gagnants.

Dans le cadre des compétitions de danse, les juges ordonnent les compétiteurs ou couples de compétiteurs selon leur performance perçue. Dans [**103**] (2003), une méthode de

Kemeny-Young légèrement modifiée est utilisée pour le classement de couples de danseurs dans le cadre de compétitions. Cette modification sert à réduire le nombre d'égalités. L'auteur défend l'utilisation de cette méthode avec des critères des systèmes électoraux et recommande d'avoir un nombre impair de juges pour les compétitions pour éviter un nombre important d'égalités.

Dans le monde des Olympiques, différentes méthodes existent pour déterminer les vainqueurs d'une compétition. M. Truchon s'est intéressé particulièrement au patinage artistique dans [**134**] (2005) et a comparé différentes méthodes d'agrégation des résultats des juges. Le choix d'une méthode peut influencer grandement le choix du vainqueur d'une compétition. L'ancienne méthode (ISU-94) utilisée en pratique avait comme base le rang médian des compétiteurs dans les classements des juges alors que la méthode récente (ISU-98) est une variante de la méthode de Copeland. Dans les deux cas, des critères additionnels sont utilisés pour briser les égalités. Truchon montre qu'il existe des cycles dans les résultats olympiques, ce qui complique le classement des compétiteurs, et compare les méthodes ISU-94, ISU-98, Borda, Copeland (voir Section 2.4.2) et Kemeny-Young. Des propriétés mathématiques de ces méthodes et le vote stratégique sont discutées. La notion de classement de Kemeny moyen est introduite : c'est un classement dans lequel les éléments sont ordonnées en fonction de leur position moyenne dans tous les permutations médianes. L'auteur suggère que la méthode de Kemeny-Young pourrait être utilisée dans ces compétitions car elle a des propriétés plus intéressantes dont celle d'être plus difficile à manipuler par un juge dans ce contexte.

Dans [**18**] (2014), Betzler et al. utilisent des données provenant de plusieurs compétitions sportives pour évaluer leur approche basée sur la réduction d'espace. Ces données proviennent des coupes du monde de skicross et du saut à ski et des saisons de la Formule 1 (courses automobiles) de 1970 à 2008. Dans la F1, pour déterminer le vainqueur d'une saison, un système à base de pointage est présentement utilisé. Les auteurs montrent que dans une saison de F1, le vainqueur officiel n'était pas le même que le vainqueur dans le consensus de Kemeny (2e officiellement). Le vainqueur officiel avait plus de points malgré que le second l'avait battu individuellement dans la majorité des courses.

3.8. Limites de l'application

Si la médiane de permutations est utilisée dans plusieurs contextes, un critère est important à souligner : la taille des permutations. Dans [**128**] (2012), des chercheurs discutent longuement de l'inapplicabilité de la méthode de Kemeny-Young dans leur cas en raison

du facteur de la taille prohibitive et des classements incomplets engendrés par l'évaluation de centaines d'athlètes du baseball. Dans leur travail, les auteurs proposent leur solution d'un algorithme de style PageRank [**105**] et d'un algorithme de style Borda [**27**] additionné d'une heuristique de recherche locale. Leur fonction objective à minimiser intègre aussi les niveaux de confiance des évaluateurs (i.e. des poids associés au permutations).

Le jeu "Thumbs-Up !" de Yahoo ! inc. [**50**] (2012) (non disponible au public, seulement à l'interne) était un jeu dans lequel les joueurs pouvaient classer des résultats d'une recherche sur le web. Pour une requête, le jeu propose des paires d'images de sites web et le joueur doit choisir le site le plus pertinent d'après lui. Plusieurs méthodes d'agrégation de ces préférences individuelles sont comparées par un score sur les résultats (Discounted cumulative gain), mais la méthode de Kemeny s'en tire mal, n'étant pas liée à ce score. Donc la nature du problème ou de la fonction à optimiser joue un rôle important dans l'applicabilité de la médiane de permutations.

Au final, le problème de la médiane de permutations peut être utilisé dans n'importe quel contexte où l'on s'intéresse à trouver un consensus sur plusieurs classements de taille raisonnable.

3.9. Stratégie de vote

Une stratégie de vote consiste à exprimer un vote qui ne reflète pas notre propre préférence, avec l'intention d'influencer le résultat du vote pour qu'il se rapproche de notre propre préférence. Une stratégie de vote est souvent faite avec une connaissance partielle ou totale des préférences des autres électeurs ce qui permet d'approximer le résultat et de choisir le vote stratégique le mieux adapté.

Comme la plupart des méthodes de votes, la méthode de Kemeny-Young n'est pas épargnée des stratégies de votes. En fait, le théorème de Gibbard-Satterthwaite dit que tout système électoral qui n'est pas dictatorial peut être manipulé [**13, 67, 115**]. La méthode de Kemeny-Young qui n'est pas dictatoriale, est donc manipulable. À titre indicatif, le système électoral qui consiste à piger un vote au hasard (lottery voting/random ballot) [**7**] n'est pas manipulable.

La Figure 3.1 illustre un exemple de stratégie de vote. La figure reprend l'exemple de la Figure 1.4 dans un cas où l'électeur #2 (représentant le vote π_2) connaissant les autres votes d'avance, peut utiliser une astuce et déclarer un vote qui n'est pas franc (le vote π_2') pour rapprocher le consensus de sa vraie préférence (qui est π_2).

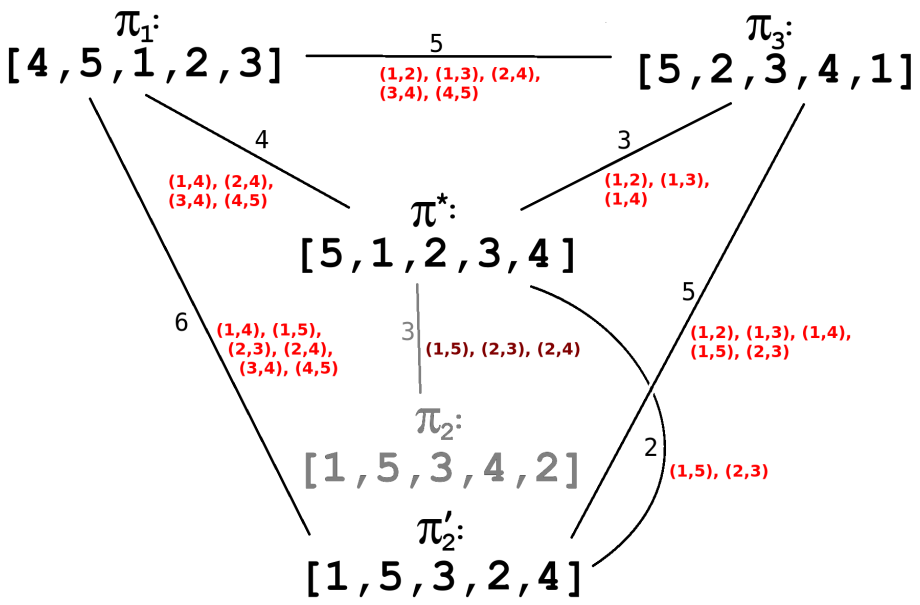


FIGURE 3.1. Exemple de stratégie de vote. La figure reprend l'exemple de la Figure 1.4. Ici, en connaissance des autres votes, l'électeur #2 change son vote pour π_2' qui diffère maintenant de sa vraie préférence personnelle π_2. En poussant le choix 4 en dernière position, il change la médiane π^* en la rapprochant de sa vraie préférence personnelle (la distance entre π^* et π_2 passe de 4 à 3). Sur chaque arête, d indique la distance de Kendall-τ entre les permutations aux sommets et les paires d'éléments en désaccords sont indiquées en rouge. Le score de Kemeny est de $d_{K\tau}(\pi_1,\pi_*) + d_{K\tau}(\pi_2,\pi_*) + d_{K\tau}(\pi_3,\pi_*) = 4 + 2 + 3 = 9$ dans ce cas.

La méthode de Kemeny-Young a été étudiée dans un contexte de votes répétés dans [24]. Dans ce contexte, plusieurs électeurs votent en fonction de leurs préférences personnelles sur plusieurs tours. Par contre, les électeurs peuvent utiliser des stratégies en changeant leur vote et ainsi influencer le résultat du vote pour le rapprocher de leur préférence personnelle. Dans ce travail, les auteurs décrivent un agent informatique (un algorithme) qui performe mieux que les agents humains dans leur stratégie. À titre d'exemple, un agent informatique consiste à choisir une permutation pour le nouveau tour de vote qui serait la permutation optimale au dernier tour de vote.

Ce chapitre a fait un tour des applications du problème de la médiane de permutations et du consensus de classements dans plusieurs domaines. Les prochains chapitres vont présenter les travaux réalisés dans le cadre de cette thèse. Ils ont trait à la réduction d'espace, aux heuristiques et aux approches exactes.

Chapitre 4

RÉDUCTION D'ESPACE POUR LE PROBLÈME DE LA MÉDIANE DE PERMUTATIONS

Résumé

Cet article journal [98], accepté dans le Journal of Discrete Applied Mathematics en avril 2018 s'inscrit dans la suite des travaux réalisés lors de ma maîtrise ; c'est en fait une version allongée de l'article [96], présenté à la conférence Conference on Algorithms and Discrete Applied Mathematics en 2016.

Dans [96] est présenté une cascade de 3 théorèmes qui permettent de déterminer l'ordre entre certaines paires d'éléments dans toutes les permutations médianes d'un ensemble de permutations. Ils sont nommés Major Order Theorems (MOT) et sont basés sur l'interférence des autres éléments entre une paire donnée d'éléments. Dans la version journal présentée ici, un cas limite des théorèmes est étudié et permet de déterminer l'ordre de nouvelles paires d'éléments dans un sous-ensemble des médianes, augmentant ainsi l'efficacité de la méthode présentée dans [96]. Une implémentation vectorielle des théorèmes y est aussi discutée. Finalement, un algorithme branch-and-bound pour résoudre des petites instances du problème y est brièvement présenté ainsi que l'avantage d'y intégrer les MOT qui réduisent l'espace de recherche. Un code en Java est disponible et libre d'utilisation à l'adresse http://www-etud.iro.umontreal.ca/~miloszro/caldam/dam_caldam.html pour permettre à tous de reproduire les résultats présentés.

Partage du travail

Dans les contributions de cet article, Robin Milosz et Sylvie Hamel ont développé et prouvé les théorèmes. Robin Milosz a implémenté en langage Java les théorèmes et le branch-and-bound, puis généré les statistiques. Tous les auteurs ont rédigé l'article.

Space reduction constraints for the median of permutations problem

Robin Milosz and Sylvie Hamel

DIRO - Université de Montréal ,
C. P. 6128 Succ. Centre-Ville, Montréal, Qc, Canada, H3C 3J7

4.1. Abstract

Given a set $\mathcal{A} \subseteq \mathcal{S}_n$ of m permutations of $\{1,2,\ldots,n\}$ and a distance function d, the **median** problem consists of finding a permutation π^* that is the "closest" of the m given permutations. Here, we study the problem under the Kendall-τ distance which counts the number of pairwise disagreements between permutations. This problem has been proved to be NP-hard when $m \geq 4$, m even. In this article, which is a full version of the conference paper [96], we investigate new theoretical properties of $\mathcal{A}$ that solve the relative order between pairs of elements in median permutations of $\mathcal{A}$, thus drastically reducing the search space of the problem. The resulting preprocessing of the problem is implemented with a Branch-and-Bound solving algorithm. We analyse its performance on randomly generated data and on real data.

4.2. Introduction

The problem of finding medians of a set of m permutations of $\{1,2,\ldots,n\}$ under the Kendall-τ distance [78, 133] is often cited in the literature as the Kemeny Score Problem [76]. In this problem m voters have to order a list of n candidates according to their preferences. The problem then consists of finding a *Kemeny consensus* : an order of the candidates that agrees the most with the order of the m voters, *i.e.*, that minimizes the sum of the disagreements.

Applications are manifold in various areas of computer science such as algorithmics [2, 44, 117, 135], artificial intelligence [18], bioinformatics [30, 40, 52], complexity theory [21], databases [59, 127] as well as in other fields such as graph theory [3, 80], information retrieval [15] and social sciences [6], to name but a few.

This problem is polynomial-time solvable for $m = 2$, has been proved to be NP-complete when $m \geq 4$, m even (first proved in [56], then corrected in [23]), but its complexity remains unknown for $m \geq 3$, m odd. In the last two decades, different exact and approximation algorithms have been derived. For finding medians of sets of small permutations, exact algorithms were described using either Branch-and-Bound techniques or linear programming with CPLEX [44, 51, 92]. For sets of bigger permutations, heuristics are needed. In 1999, a greedy algorithm with approximation factor 2 was described in [39] and five years later, a better one was developed in [51]. Then, a randomized algorithm with approximation factor 11/7 [3] and a deterministic one with approximation factor 8/5 [135] were designed. In 2007, a PTAS result was obtained [80] and some years later, different fixed-parameter algorithms have been described [20, 22, 75, 104, 123]. A comparative study of most of these approaches can be found in [6].

Solving the median problem may also be seen as solving the order of all the $\frac{n(n-1)}{2}$ possible pairs of elements in a median. Other theoretical approaches working in that direction and aiming at reducing the search space for this problem have also been developed. In [26], a theorem targeting pairwise ordering was proposed along with some constraints about adjacent elements of a median. A data reduction was proposed in [18] where a "non-dirty" element can be ordered with respect to all other elements in a median, splitting the ordering problem into two smaller ones.

In this present work, we are interested in the theoretical perspective of the problem, investigating properties of an instance that can partly resolve and accelerate computation. We derive necessary conditions for setting the order of appearance of pairs of elements in a median, building constraints that drastically reduce the search space for this median. Note that this article is an extended version of the conference paper [96]. In this version, all sketches of proofs are replaced by full proofs, new sections are added (Sections 4.3.3, 4.4 and 5.2) and the efficiency of our approach is tested on larger randomized permutations (Section 5.1) and on real data (Section 5.2).

This article is organized as follows : after introducing the basic definitions and notations in Section 2, we resume, in Section 3, some related previous works aiming at reducing the search space for the medians. Section 4 presents our approach, while Section 5 presents its results on uniformly generated random sets of permutations and on real data. Section 5 also compares our approach with the ones discussed in Section 3. Finally, we conclude and give some future directions for this work in Section 6.

4.3. Median of permutation : definitions and notations

A **permutation** π is a bijection of $[n] = \{1,2\ldots,n\}$ onto itself. The set of all permutations of $[n]$ is denoted $\mathcal{S}_n$. As usual we denote a permutation π of $[n]$ as $\pi = \pi_1\pi_2\ldots\pi_n$. Let $\mathcal{A} \subset \mathcal{S}_n$ be a set of permutations of $[n]$, we will denote its cardinality by $\#\mathcal{A}$.

The **Kendall-τ distance**, denoted d_{KT}, counts the number of pairwise disagreements between two permutations and can be defined formally as follows : For permutations π and σ of $[n]$, we have that

$$d_{KT}(\pi,\sigma) = \#\{(i,j)|i < j \text{ and } [(\pi[i] < \pi[j] \text{ and } \sigma[i] > \sigma[j]) \text{ or } (\pi[i] > \pi[j] \text{ and } \sigma[i] < \sigma[j])]\},$$

where $\pi[i]$ denotes the position of integer i in permutation π.

Given any set of permutations $\mathcal{A} \subseteq \mathcal{S}_n$ and a permutation π, we have

$$d_{KT}(\pi,\mathcal{A}) = \sum_{\sigma\in\mathcal{A}} d_{KT}(\pi,\sigma).$$

The **problem of finding a median of $\mathcal{A}$ under the Kendall-τ distance** can be stated formally as follows : Given $\mathcal{A} \subseteq \mathcal{S}_n$, we want to find a permutation π^* of $[n]$ such that $d_{KT}(\pi^*,\mathcal{A}) \leq d_{KT}(\pi,\mathcal{A})$, $\forall\ \pi \in \mathcal{S}_n$. Note that a set $\mathcal{A}$ can have more than one median.

4.4. Previous approaches

When dealing with permutations, searching through the whole set of permutations $[n]$ quickly becomes impossible since there are $n!$ such permutations. To be able to find exact medians for sets of "big" permutations, we need to reduce the search space so that the computation will take place in a reasonable time.

Here, given a set of permutations $\mathcal{A} \subseteq \mathcal{S}_n$, we present two previous approaches that reduce the search space by discarding non-relevant permutations.

4.4.1. Data reduction with non-dirty candidates

In [18] a **non-dirty pair** of candidates according to a certain threshold $s \in [0,1]$ is a pair of elements (a,b), $a,b \in [n]$, which respect the following property : either a is before b (element a is placed to the left of element b) in a ratio of s or more in the starting set of permutations $\mathcal{A}$, or b is before a in a ratio of s or more in the starting set $\mathcal{A}$. A **non-dirty candidate** is an element which forms a non-dirty pair with every other element of $[n]$ according to the threshold s.

It has been proven in [18] that with $s = 3/4$, elements of a median permutation are necessarily ordered relatively to a non-dirty candidate in the majority order. In other words, a non-dirty candidate separates the median permutation such as all the elements that are before it are to its left and all the elements that are not are to its right. In what follows, we will refer to this result as the *3/4 majority rule*.

Having such a non-dirty candidate allows to cut the instance of the problem in two sub-instances, on which the approach can be re-applied. It also allows the parallelization of the problem because the two sub-instances are independent. Betzler's *et al.* approach is strong on sets of permutations derived from real data, since, in those cases, the permutations are often really close to each other, greatly increasing the probability of finding non-dirty candidates.

The drawback is that non-dirty candidates are rare on uniformly generated random sets of permutations, like shown in Table 4. V and discussed in Section 4.6.

4.4.2. Always theorem

In [26] a pairwise ordering theorem was described for a pair of elements (a,b) such that if a is before b in all permutations of $\mathcal{A}$ then a is necessarily before b in any median permutation. This theorem allows to set the relative order in pairs of elements that respect the property of being always at the right or always at the left of the other element. We will refer to this theorem as the *always theorem*.

The drawback is that the efficiency of this theorem, in terms of the proportion of pairs that it will ordered, for a set of uniformly distributed random permutations is greatly affected by the number of permutations m in our given set $\mathcal{A}$. The probability of having such a pair is $p = \frac{1}{2^{m-1}}$ because it has to keep the same order in each of the m permutations and there are two possible orders $a \prec b$ (a before b) and $b \prec a$ (b before a). For three permutations, $p = 25\%$, for four permutations $p = 12.5\%$ and so on (See Table 4. I).

4.5. Our approach

We were inspired by the two approaches discussed in Section 3, with the aim of building ordering constraints for pairs of elements in a median. We ask ourselves if there was a data reduction which was less restrictive than the *3/4 majority rule* but more general than the *always theorem*.

4.5.1. Existence of a majority bound ?

Given that the *always theorem* guarantees the order of a pair of elements in a median, our first idea was to find a %-majority bound, where having an element before another one in at least $p\%$ of the permutations of $\mathcal{A}$ would guarantee the order of this pair of elements in a median of $\mathcal{A}$. Unfortunately, no such bound exists.

Proposition 1. *For any given bound s, $0.5 < s < 1$, it is always possible to construct a set $\mathcal{A}$ in which the proportion $p\%$ of permutations favoring element i to element j (the major order) is at least s, $s \leq p\% < 1$, but for which the order of the pair (i,j) in any median of $\mathcal{A}$ contradicts this majority order.*

Proof. Let $\mathcal{A}$ be the following set of permutations of $[n]$:

$$\mathcal{A} = \{\sigma_1,\sigma_2,\ldots,\sigma_a,\gamma,\pi_1,\pi_2,\ldots,\pi_a\},$$

where each σ_i, $1 \leq i \leq a$, is a different permutation beginning with element 1 followed by element 2 ; each π_i, $1 \leq i \leq a$, is a different permutation ending with element 1 followed by element 2, and γ is a permutation beginning by element 2 and ending by element 1 (in γ elements 1 and 2 are thus separated by a set K of k elements). Note : $n = k + 2$.

So, in $2a$ permutations, element 1 is before element 2, and in only one, element 2 is before element 1. Even if the majority order for this pair of elements is 1 before 2, we can show that for an arbitrary a, if $k > 2a - 1$ then the median of $\mathcal{A}$ will be of the form $2K1$, more precisely the median will be $2\alpha1$ where 2 is before 1 and where α is the optimal permutation of the elements of set K (i.e. $\pi^* = 2\alpha1$ is a median of $\mathcal{A}$). We then choose a minimal a such that $\frac{2a}{2a+1} \geq s$ and $k = 2a$ to complete the construction.

First, we observe that the cost of every element of K is identical in relation to 1 and 2 simply because the elements of K are always grouped together in the permutations of $\mathcal{A}$.

Let us define an optimal arrangement of a given subset S of $[n]$ as a permutation of the elements of S that minimize the Kendall-τ distance to $\mathcal{A}$, where only the elements of S are considered in each permutation of $\mathcal{A}$.

Second, we can state that there exist at least one optimal arrangement for the elements of K based on the order they were placed in the permutations of $\mathcal{A}$. Let α be any optimal arrangement for the elements of K and C_α, its associated cost.

The two previous points lead to this observation : a median permutation of $\mathcal{A}$ will have the elements of K arranged in an optimal arrangement α. This is because of the fact that if a supposed median permutation has the elements of K arranged in a non-optimal way, we could make a bunch of swaps between those elements of K, omitting 1 and 2, so

that their relative order will become an optimal arrangement, thus lowering the cost of the permutation.

Now, a median of $\mathcal{A}$ can either be of the form $K_1 1 K_2 2 K_3$ or of the form $K_1 2 K_2 1 K_3$, where the K_i are possibly empty subsets of K, such that $\bigcup_{i=1}^{3} K_i = K$, $K_i \bigcap K_j = \varnothing$ if $i \neq j$, $|K_i| = k_i$ and $k_1 + k_2 + k_3 = k$. The order of the elements of K is known and follows α, so the only variables to be set are the numbers of elements to be placed in K_1, K_2 and K_3. In other words, we have α and we need to place in 1 and 2 in an optimal way to attain the median.

If our median is of the form $K_1 1 K_2 2 K_3$ we have

$$\begin{aligned} d_{KT}(K_1 1 K_2 2 K_3, \mathcal{A}) &= \overbrace{a(2k_1 + k_2)}^{1} + \overbrace{a(k_2 + 2k_3)}^{2} + \overbrace{1(1 + k_1 + 2k_2 + k_3)}^{3} + C_\alpha \\ &= 2ak + 1 + k_1 + 2k_2 + k_3 + C_\alpha, \end{aligned}$$

where terms 1, 2 and 3 capture the distance with respectively the permutations σ_i, $1 \leq i \leq a$, the permutations π_i, $1 \leq i \leq a$, and the permutation γ of $\mathcal{A}$, for the pairs (1,x), (2,x) and (1,2), $x \in K$. The cost for the pairs (x_1,x_2), x_1,$x_2 \in K$, is counted in C_α.

Following the same logic, if our median is of the form $K_1 2 K_2 1 K_3$, we have

$$\begin{aligned} d_{KT}(K_1 2 K_2 1 K_3, \mathcal{A}) &= a(2k_1 + k_2 + 1) + a(k_2 + 2k_3 + 1) + 1(k_1 + k_3) + C_\alpha \\ &= 2ak + 2a + k_1 + k_3 + C_\alpha. \end{aligned}$$

If we try to minimize $d_{KT}(K_1 1 K_2 2 K_3, \mathcal{A})$, we will have to put all the k elements of K in either K_1 or K_3 because putting them in K_2 will have a doubled penalty for the cost. Doing so, the minimal cost for $K_1 1 K_2 2 K_3$ is $2ak + 1 + k + C_\alpha$. Now, if we try to minimize $d_{KT}(K_1 2 K_2 1 K_3, \mathcal{A})$, we will have to put all the k elements of K in K_2 because putting them in either K_1 or K_3 will give a penalty whereas K_2 is "silent" in the cost function. So, the minimal cost for $K_1 2 K_2 1 K_3$ is $2ak + 2a + C_\alpha$. Since it is stated in the proposition that $k > 2a - 1$, the minimal cost of $K_1 1 K_2 2 K_3$ is strictly greater than the minimal cost of $K_1 2 K_2 1 K_3$ ($2ak + 1 + k + C_\alpha > 2ak + 2a + C_\alpha$).

This gives us that median permutations π^* of $\mathcal{A}$ are of the general form $K_1 2 K_2 1 K_3$, where K_1 and K_3 are empty. Thus π^* is of the form $2K1$ or more precisely $\pi^* = 2\alpha 1$, where α be any optimal arrangement for the elements of K with respect to $\mathcal{A}$. Element 2 is before element 1 in the median, regardless of the large proportion of permutations of $\mathcal{A}$ favoring 1 to 2. This finishes the proof of non-existence of a %-majority bound. ■

4.5.2. Major Order Theorem

Since nothing can be derived from a majority ordering, another idea comes from the observation that two elements that are close enough in all permutations of $\mathcal{A}$ have the tendency to be placed in their major order (the relative order in which they appear the most in the permutations of $\mathcal{A}$), in any median of $\mathcal{A}$. The intuition is that there is less "interference" caused by other elements.

The total absence of "interference" between two elements is found when they are adjacent in all permutations. In that case, we can consider them as one heavier element and their relative order in a median permutation will clearly be the major order.

So, what happens if we limit the "interference" between two elements ? Can we then have an extension of the *always theorem* ? The answer is yes and is given by our Major Order Theorem below (Theorem 1). But first, we need more definitions and notations.

Definition 1. *Let $\mathcal{A} \subseteq \mathcal{S}_n$ be a set of m permutations of $[n]$. Let us build, for each pair of elements (i,j), $1 \leq i < j \leq n$, two multisets $E_{ij}(\mathcal{A})$ and $E_{ji}(\mathcal{A})$.*

Multiset $E_{ij}(\mathcal{A})$ (resp. $E_{ji}(\mathcal{A})$) contains all the elements present between elements i and j in all permutations of $\mathcal{A}$, where element i is positioned before (resp. after) element j, denoted $i \prec j$ (resp. $j \prec i$). Mathematically, we have

$$E_{ij}(\mathcal{A}) = \bigcup_{\substack{\forall \pi \in \mathcal{A}, \\ k \in [n]}} \{k \,|\, \pi[i] < \pi[k] < \pi[j]\},$$

and

$$E_{ji}(\mathcal{A}) = \bigcup_{\substack{\forall \pi \in \mathcal{A}, \\ k \in [n]}} \{k \,|\, \pi[j] < \pi[k] < \pi[i]\}.$$

When there are no ambiguities, we simply denotes $E_{ij}(\mathcal{A})$ by E_{ij}.

To keep track of the number of permutations in $\mathcal{A}$, that have a certain order between two elements, let us introduce the left/right distance matrices L and R.

Definition 2. *a) Let $L(\mathcal{A})$, or simply L, when there are no ambiguities, be the* **left distance matrix** *of $\mathcal{A}$, where $L_{ij}(\mathcal{A})$ denotes the number of permutations of $\mathcal{A}$ having element i to the left of the element j.*

b) Symmetrically, let $R(\mathcal{A})$, or simply R, be the **right distance matrix** *of $\mathcal{A}$, where $R_{ij}(\mathcal{A})$ denotes the number of permutations of $\mathcal{A}$ having element i to the right of the element j. Obviously, $L_{ij} + R_{ij} = m$, the number of permutations in $\mathcal{A}$ and $L_{ij} = R_{ji}$. Note that $d_{KT}(\pi, \mathcal{A}) = \sum_{\substack{i,j \\ \pi[i]<\pi[j]}} R_{ij}(\mathcal{A})$.*

c) For $S \subseteq [n]$, we define L_{xS} as $L_{xS} = \sum_{s \in S} L_{xs}$, i.e. the total number of times x is to the left of an element of S in all permutations of $\mathcal{A}$. $R_{xS} = \sum_{s \in S} R_{xs}$ is defined symmetrically.

We are now almost ready to state our Major Order Theorem but first let us formally define the major order between elements.

Definition 3. *We say that the* **major order** *between elements i and j is $i \prec j$ (resp. $j \prec i$) if $L_{ij} > R_{ij}$ (resp. $R_{ij} > L_{ij}$), the* **minor order** *is then $j \prec i$ (resp. $i \prec j$). We use δ_{ij} to denote the difference between the major and minor order of two elements i and j, $\delta_{ij} = |L_{ij} - R_{ij}| = |R_{ij} - L_{ij}| = |L_{ji} - R_{ji}| = \delta_{ji}$. If $R_{ij} = L_{ij}$, there are no major nor minor order.*

Theorem 1. (Major Order Theorem 1.0) *Let $\mathcal{A} \subseteq \mathcal{S}_n$ be a set of permutations of $[n]$. For a pair of elements (i,j), $1 \leq i < j \leq n$, if $i \prec j$ (resp. $j \prec i$) is their major order and $\delta_{ij} > \#E_{ji}$ (resp. $\delta_{ij} > \#E_{ij}$) then this major order is conserved in all medians π^* of $\mathcal{A}$.*

Proof. Suppose, without lost of generality, that for a pair of elements (i,j) the major order is $i \prec j$ and $\delta_{ij} > \#E_{ji}$. By contradiction, suppose that we have a median permutation π^* for $\mathcal{A}$ in which i and j are in their minor order $j \prec i$. Let $\pi^* = B\, j\, K\, i\, A$ be such a median, where B, K and A are the sets of elements found before (B), in between (kernel - K) and after (A) elements i and j.

The contribution of element i to the Kendall-τ distance $d_{KT}(\pi^*,\mathcal{A})$ is

$$\underset{\substack{\#\text{ of times } i \text{ is left to } b \in B,\\ \text{in all } \sigma \in \mathcal{A}}}{L_{iB}} + \underset{\substack{\#\text{ of times } i \text{ is left to } k \in K,\\ \text{in all } \sigma \in \mathcal{A}}}{L_{iK}} + \underset{\substack{\#\text{ of times } i \text{ is right to } a \in A,\\ \text{in all } \sigma \in \mathcal{A}}}{R_{iA}} + L_{ij}.$$

Similarly, the contribution of element j to $d_{KT}(\pi^*,\mathcal{A})$ is $L_{jB}+R_{jK}+R_{jA}+L_{ij}$. We show that for either $\sigma^* = B\, i\, j\, K\, A$ or $\sigma^* = B\, K\, i\, j\, A$, we have $d_{KT}(\sigma^*,\mathcal{A}) < d_{KT}(\pi^*,\mathcal{A})$ contradicting our choice of median and, at the same time, our choice of ordering for the pair (i,j).

We will investigate interactions between i, j and elements of the set K, since the elements of the sets A and B stay in the same relative order with i and j in π^* and either choice of σ^*. (Elements in A (resp. B) will always be after (resp. before) elements i and j).

The cost of the interaction of i with K and of j with K is respectively L_{iK} and R_{jK} in the supposed median π^*. Thus, there are two possible cases : either $L_{iK} \leq R_{jK}$ or $L_{iK} > R_{jK}$.

Case 1 : $\mathbf{L_{iK} \leq R_{jK}}$

For this case, let $\sigma^* = B\,K\,i\,j\,A$, i.e. we moved element j to the immediate right of element i in our supposed median $\pi^* = B\,j\,K\,i\,A$. In this case, we have

$$\begin{aligned}
d_{KT}(\sigma^*,\mathcal{A}) - d_{KT}(\pi^*,\mathcal{A}) &\overset{(\diamond_1)}{=} (L_{jB} + L_{jK} + R_{jA} + R_{ij}) - (L_{jB} + R_{jK} + R_{jA} + L_{ij}) \\
&= (L_{jK} + R_{ij}) - (R_{jK} + L_{ij}) \\
&= L_{jK} - R_{jK} + -(L_{ij} - R_{ij}) \\
&\overset{(\square_1)}{=} L_{jK} - R_{jK} - \delta_{ij} \\
&\overset{(*_1)}{\leq} L_{iK} + \#E_{ji} - R_{jK} - \delta_{ij} \\
&\overset{Case\,1}{\leq} R_{jK} + \#E_{ji} - R_{jK} - \delta_{ij} \\
&\leq \#E_{ji} - \delta_{ij} \\
&< 0.
\end{aligned}$$

Equality $(\diamond_1)$ is obtained by taking into account only the contribution of the element that changes position between π^* and σ^*, i.e. element j, since the contribution of the other elements will cancel each other out. Equality $(\square_1)$ takes into account the fact that, in the permutations of $\mathcal{A}$, the major order for elements i and j is $i \prec j$, implying that $L_{ij} > R_{ij}$. As for inequality $(*_1)$, note that $L_{jK} \leq L_{iK} + \#E_{ji}$ since for an element $k \in K$, we will add one to L_{jk} iff j is to the left of k in a permutation of $\mathcal{A}$. In this same permutation, either i is also to the left of k (captured by adding one to L_{ik}) or to the right, in which case, k is an element of E_{ji}. The last inequality follows from the initial condition of the theorem.

Thus, moving j after i gives us a permutation σ^* which is closer to the set $\mathcal{A}$ contradicting our choice of π^* as a median of $\mathcal{A}$.

Case 2 : $\mathbf{L_{iK} > R_{jK}}$

For this case, let $\sigma^* = B\,i\,j\,K\,A$, i.e. we moved element i to the immediate left of element j in our supposed median $\pi^* = B\,j\,K\,i\,A$. In this case, we have

$$\begin{aligned}
d_{KT}(\sigma^*,\mathcal{A}) - d_{KT}(\pi^*,\mathcal{A}) &\overset{(\diamond_2)}{=} (L_{iB} + R_{iK} + R_{iA} + R_{ij}) - (L_{iB} + L_{iK} + R_{iA} + L_{ij}) \\
&= (R_{iK} + R_{ij}) - (L_{iK} + L_{ij}) \\
&= R_{iK} - L_{iK} + -(L_{ij} - R_{ij}) \\
&= R_{iK} - L_{iK} - \delta_{ij} \\
&\overset{(*_2)}{\leq} R_{jK} + \#E_{ji} - L_{iK} - \delta_{ij}
\end{aligned}$$

$$\begin{aligned} &\overset{Case\,2}{<} && L_{iK} + \#E_{ji} - L_{iK} - \delta_{ij} \\ &\leq && \#E_{ji} - \delta_{ij} \\ &< && 0. \end{aligned}$$

Equality $(\diamond_2)$ is obtained by taking into account only the contribution of the element that changes position between π^* and σ^*, i.e. element i. As for inequality $(*_2)$, it follows, by symmetry, from $(*_1)$, i.e. $R_{iK} \leq R_{jK} + \#E_{ji}$. Again, the last inequality follows from the initial condition of the theorem.

In each of the two cases, we were able to find a permutation σ^*, such that $d_{KT}(\sigma^*,\mathcal{A}) < d_{KT}(\pi^*,\mathcal{A})$, contradicting our choice of median π^* and our choice of ordering for the pair of elements (i,j). Consequently, i and j can only be placed in their major order $i \prec j$ in a median permutation if the conditions are fulfilled. ■

4.5.3. Refined versions of the Major Order Theorem

We tested our Major Order Theorem 1.0 on randomly generated sets of permutations (see Table 4. I) and saw that its efficiency, in terms of the number of pairs of elements ordered, was not that much better than the *always theorem*, as n grows. In order to increase the efficiency of our theorem, we needed to find a method to reduce the size of our multisets E_{ij} and E_{ji}.

4.5.3.1. *Refined version 1.*

We observed that the presence of an element k in both multisets E_{ij} and E_{ji} cancels its impact on the ordering of the pair of elements (i,j). This leads to a 2.0 version of our Major Order Theorem presented below.

Definition 4. *Let $\mathcal{A} \subseteq \mathcal{S}_n$ be a set of m permutations of $[n]$. For multisets $E_{ii}(\mathcal{A})$ and $E_{ji}(\mathcal{A})$ defined in Definition 1, let $E'_{ij}(\mathcal{A})$ and $E'_{ji}(\mathcal{A})$ be those new multisets defined by $E'_{ij}(\mathcal{A}) = E_{ij}(\mathcal{A}) \setminus E_{ji}(\mathcal{A})$ and $E'_{ji}(\mathcal{A}) = E_{ji}(\mathcal{A}) \setminus E_{ij}(\mathcal{A})$.*

Theorem 2. (Major Order Theorem 2.0) *Let $\mathcal{A} \subseteq \mathcal{S}_n$ be a set of permutations of $[n]$. For a pair of elements (i,j), $1 \leq i < j \leq n$, if $i \prec j$ (resp. $j \prec i$) is their major order and $\delta_{ij} > \#E'_{ji}$ (resp. $\delta_{ij} > \#E'_{ij}$) then this major order is conserved in all medians π^* of $\mathcal{A}$.*

Proof. We show that in Cases 1 and 2 of the proof of Theorem 1, $\#E_{ji}$ can be replaced by $\#E'_{ji}$, which concludes the proof.

To show that we only need to show that inequality $(*_1)$ can be replaced by inequality $L_{jK} \leq L_{iK} + \#E'_{ji}$.

To do so, let us introduce multisets $E_{ji}\{K\}$ and $E_{ij}\{K\}$ which are the equivalents of E_{ji} and E_{ij} restricted to the subset of elements $K \subseteq [n]$.

We have

$$\begin{aligned}
\underset{\substack{\#\text{ of times } j \prec k \in K,\\ \text{in all } \sigma \in \mathcal{A}}}{L_{jK}} &= \underset{\substack{\#\text{ of times } i \prec k \in K,\\ \text{in all } \sigma \in \mathcal{A}}}{L_{iK}} + \underset{\substack{\#\text{ of times } j \prec k \prec i,\\ k \in K, \text{ in all } \sigma \in \mathcal{A}}}{\#E_{ji}\{K\}} - \underset{\substack{\#\text{ of times } i \prec k \prec j,\\ k \in K, \text{ in all } \sigma \in \mathcal{A}}}{\#E_{ij}\{K\}} \\
&\overset{(\triangle_1)}{\leq} L_{iK} + \#(E_{ji}\{K\} \setminus E_{ij}\{K\}) \\
&\overset{(\square_1)}{\leq} L_{iK} + \#E'_{ji}\{K\}, \\
&\overset{(\circ_1)}{\leq} L_{iK} + \#E'_{ji},
\end{aligned}$$

where $(\triangle_1)$ follows from the fact that $\#E_{ji}(k) - \#E_{ij}(k) \leq \#(E_{ji}(k) \setminus E_{ij}(k))$, $(\square_1)$ comes from the definition of E'_{ji} (Definition 4) and $(\circ_1)$ is a consequence of $\#E'_{ji} \geq \#E'_{ji}\{K\}$, since E'_{ji} contains at least all the elements of $E'_{ji}\{K\}$. ■

This version 2.0 of our Major Order Theorem was also tested on randomly generated sets of permutations (see Table 4. I) and its efficiency, discussed in more details in Section 5, is much better than our 1.0 version even as n grows.

4.5.3.2. *Refined version 2.*

Given a set of permutations $\mathcal{A}$, Theorem 2 gives us a set of ordering constraints for some pairs of elements in a median of $\mathcal{A}$. We can use this set to extend the reach of the theorem by applying a second filter over the multisets E'_{ij}, E'_{ji}, in the following way : While investigating a pair of elements (i,j), if we previously found a pair of constraints related to an element k, ($k \prec i$ and $k \prec j$) or ($i \prec k$ and $j \prec k$), then this element k cannot be found in between i and j in a median permutation. In the proof of Theorem 1, with $\pi* = BjKiA$, we only need the constraint $k \prec j$ (or $i \prec k$) to know that element k is not in the set K. Thus, when one of these constraints is found for an element k, its contribution to L_{iK}, L_{jK}, R_{iK} and R_{jK} is null and so k can be removed from E'_{ij} and E'_{ji}.

In this way, we trim the multisets E'_{ij} and E'_{ji} by taking out all copies of elements that have been proved, by our ordering constraints, to be to the right, or to the left, of both i and j. Let E''^{1}_{ij} and E''^{1}_{ji} be these new trimmed multisets, obtained after this first iteration. We continue applying the same theorem but upgrading the $\delta_{ij} > \#E'_{ji}$ condition to $\delta_{ij} > \#E''^{1}_{ji}$. We iterate this process, obtaining at iteration t, new trimmed sets E''^{t}_{ij} and E''^{t}_{ji}. We stop the process when an iteration does not find any new constraint. This gives us the following refined version of Theorem 2 :

Theorem 3. (Major Order Theorem 3.0) *Let $\mathcal{A} \subseteq \mathcal{S}_n$ be a set of permutations of $[n]$. For a pair of elements (i,j), $1 \leq i < j \leq n$, if $i \prec j$ (resp. $j \prec i$) is their major order and $\delta_{ij} > \#E''^{t}_{ji}$, $t > 0 \in \mathbb{N}$ (resp. $\delta_{ji} > \#E''^{t}_{ij}$) then this major order is conserved in all medians π^* of $\mathcal{A}$.* ■

4.5.3.3. *Adding equality*

In Theorem 3, we have the initial condition $\delta_{ij} > \#E''^{t}_{ji}$. What happens if we consider $\delta_{ij} \geq \#E''^{t}_{ji}$? This means that given a supposed median permutation $\pi* = BjKiA$ we can assert that $\sigma^* = BKijA$ or $\sigma^* = BijKA$ will have a score $d_{KT}(\sigma^*,\mathcal{A}) \leq d_{KT}(\pi^*,\mathcal{A})$. Here, we cannot state that all median permutations will have $i \prec j$ but we can state that at least one median permutation will have $i \prec j$. This also includes the case where $\delta_{ij} = 0$ i.e. when the major order does not exist between two elements (as it is possible with an even number of permutations in $\mathcal{A}$).

Theorem 4. (Major Order Theorem 3.0 with equality - MOT3.0e)
Let $\mathcal{A} \subseteq \mathcal{S}_n$ be a set of permutations of $[n]$. For a pair of elements (i,j), $1 \leq i < j \leq n$, if $i \prec j$ (resp. $j \prec i$) is their major order and $\delta_{ij} \geq \#E''^{t}_{ji}$, $t > 0 \in \mathbb{N}$ (resp. $\delta_{ji} \geq \#E''^{t}_{ij}$) then this major order is conserved in at least one median π^* of $\mathcal{A}$.

Proof. The same as for Theorem 3 changing the inequality $\delta_{ij} > \#E''^{t}_{ji}$ for the inequality $\delta_{ij} \geq \#E''^{t}_{ji}$. ■

One problem arises from allowing this possibility for equality : the consistency of our constraints. In other words, we need to be careful when we add a constraint found in the equality case so that we do not build an inconsistent set of constraints as the constraints found by enabling the equality case may arise from different median permutations of $\mathcal{A}$.

Example 1. *Let $\mathcal{A}$ = {1234,2341,3412,4123}. For this set $\mathcal{A}$, Theorem 4 (MOT3.0e) is applicable for each of the pairs of elements :* (1,2), (2,3), (3,4) *and* (4,1). *Thus, all those pairs taken individually are eligible to be constraints but it is not the case of all pairs*

put together, which clearly are contradictory with respect to transitivity. To assure the consistency of the set of constraints, we need to take a transitive closure of the constraints set every time we add a constraint using the equal case. In the previous special case, after finding $1 \prec 2$, $2 \prec 3$ *and* $3 \prec 4$ *we would have* $1 \prec 4$ *by transitivity, forbidding the investigation of the pair* (4,1). *In this case, we would have found a median permutation :* 1234 *but not all the medians of* $\mathcal{A}$. *The fact that we did find that the pair* (4,1) *also satisfy Theorem 4 means that in another median of* $\mathcal{A}$, $4 \prec 1$. *(In fact, in this case, the set of medians of* $\mathcal{A}$ *is equal to the set* $\mathcal{A}$ *itself, so permutation* 4123 *is also a median of* $\mathcal{A}$, *satisfying* $4 \prec 1$*).*

Restricting our effort to finding only one median of $\mathcal{A}$ is remarkably rewarding as the efficiency of our Major Order Theorem is greatly improved. As we can see in Table 4. III, the Major Order Theorem 3.0 with equality (MOT3.0e) solves the relative order of a majority of pairs in smaller to medium instances, improving greatly on the efficiency of MOT3.0. Also, in a vast majority of cases, MOT3.0e is finding all $3\backslash 4$ Majority Rule constraints, and almost all of them, in the other cases (see Table 4. VI).

4.5.4. Implementation - Vectorial version of the Major Order Theorem

To compute easily the set of constraints arising from our Major Order Theorem, we describe vectorial versions of them that we use in our implementation. This vector approach does not use a specific data structure, uses less memory and is much faster which allows us to compute the median of sets of larger permutations. Let us first set some notations.

Notations : Let $\overrightarrow{L_x}$ be the vector of the left distance matrix L of Definition 2, associated to the element x. Note that $\overrightarrow{L_x}[y] = L_{xy}$. $\overrightarrow{R_x}$ is defined symmetrically.
Now, let us define

$$\Delta_{ij} = \max_{\substack{K \subseteq \{1 \dots n\} \\ i,j \notin K}} \{R_{iK} - R_{jK}\}.$$

Then, $R_{iK} - R_{jK} \leq \Delta_{ij}$ and $R_{iK} \leq R_{jK} + \Delta_{ij}$. Also, $L_{jK} \leq L_{iK} + \Delta_{ij}$. Intuitively, Δ_{ij} represent the worst "interference" between elements i and j.

Vectorially, we have

$$\Delta_{ij} = \sum_{\substack{k=0 \\ k \neq i,j \\ \overrightarrow{R_i} - \overrightarrow{R_j}[k] > 0}}^{n} \overrightarrow{R_i} - \overrightarrow{R_j}[k].$$

We observe that $\Delta_{ij} = \#E'_{ji}$ if the major order is $i \prec j$. $\#E'_{ji}$ is the number of occurrences of elements found between j and i in permutations where $j \prec i$, minus the occurrences of those elements when found between i and j in permutations where $i \prec j$ (so if an element is more often found between i and j than between j and i, it will not be in E'_{ji}). Δ_{ij} sums over all positive values of $\overrightarrow{R_i} - \overrightarrow{R_j}$ not in position i or j. If an element k is to the left (resp. to the right) of both i and j in a permutation of $\mathcal{A}$ then it will not affect the difference $\overrightarrow{R_i} - \overrightarrow{R_j}[k]$. The only case when $\overrightarrow{R_i}[k] - \overrightarrow{R_j}[k] > 0$, is when k is in between j and i ($j \prec k \prec i$) more times than in between i and j($i \prec k \prec j$) in permutations of $\mathcal{A}$. That difference $(\#(j \prec k \prec i) - \#(i \prec k \prec j))$ is exactly what is counted by $\#E'_{ji}(\mathcal{A})$. Thus, $\Delta_{ij} = \#E'_{ji}$.

Now to obtain a vectorial equivalence to our sets E''^{t}_{ji}, we have to remove from Δ_{ij} any element ℓ for which we have already found a constraint $i \prec \ell$ or $\ell \prec j$ (so ℓ in not in between j and i). To do so, we only have to define sets Δ'^{t}_{ij} as

$$\Delta'^{t}_{ij} = \sum_{\substack{k=1 \\ k \neq i,j \\ k<j \text{ or } i<k \notin \mathcal{C}_{t-1} \\ \overrightarrow{R_i} - \overrightarrow{R_j}[k] > 0}}^{n} \overrightarrow{R_i} - \overrightarrow{R_j}[k],$$

where $\mathcal{C}_{t-1}$ is the set of constraints from iteration $t-1$. Note that $\mathcal{C}_0$ contains the constraints found by our Theorem 2. Δ'^{t}_{ij} is then equal to $\#E''^{t}_{ji}$ from the previous Theorem 3 and 4.

4.6. Efficiency of our approach

In this section, we will present the efficiency of our approach on randomly generated data and on real data. We also compare it to the previous approaches briefly described in Section 3. In what follows, n will represent the number of elements in our permutations and m the size of the set of permutations considered.

We will base our efficiency statistics on the proportion of solved ordering of pairs of elements. (For permutations of $[n]$, there are $\frac{n(n-1)}{2}$ pairs to order.)

Note that for efficiency and theoretical concerns, after applying any theorem on a set of permutations $\mathcal{A}$, we apply the transitive closure on the sets of constraints found. Since, if we know that an element i is to the left of j and element j is to the left of k, then we also know that element i will be left of k. (The *always theorem* is always transitive closed but it is not the case for our Major Order Theorems.)

4.6.1. Efficiency on randomly generated data

We evaluated our approach on uniformly generated random permutation sets. For each pair m, n, statistics were calculated over 10 000 (bigger n) to 1 000 000 (smaller n) instances (see Table 4. IV for the exact number of instances for each n). Fisher-Yates shuffle also known as Knuth shuffle [**64, 82**] was used to create random permutations which guarantees that the generation is uniform (every permutation is equally likely).

Table 4. I shows the efficiency of the *always theorem* and versions 1.0, 2.0 and 3.0 of our Major Order Theorems on sets of permutations of $[n]$, when $n = 15$ or $n = 30$.

	$n = 15$				$n = 30$			
m	Always Thm	Maj. Order Thm 1.0	Maj. Order Thm 2.0	Maj. Order Thm 3.0	Always Thm	Maj. Order Thm 1.0	Maj. Order Thm 2.0	Maj. Order Thm 3.0
3	0.2496	0.3603	0.4981	0.6345	0.2498	0.3055	0.3962	0.5065
4	0.1248	0.2595	0.4711	0.5201	0.1248	0.1937	0.3658	0.4123
5	0.0626	0.1928	0.4648	0.5813	0.0626	0.1272	0.3478	0.4036
10	0.0020	0.0530	0.4435	0.5173	0.0020	0.0199	0.3194	0.3619
11	0.0010	0.0419	0.4478	0.5478	0.0010	0.0139	0.3174	0.3609
12	0.0005	0.0328	0.4418	0.5182	0.0005	0.0098	0.3149	0.3558
13	0.0002	0.0261	0.4457	0.5450	0.0002	0.0070	0.3147	0.3570
14	0.0001	0.0208	0.4415	0.5199	0.0001	0.0050	0.3133	0.3535
15	0.0001	0.0165	0.4453	0.5445	0.0001	0.0036	0.3130	0.3545
20	0	0.0054	0.4415	0.5248	0	0.0007	0.3096	0.3485

TABLEAU 4. I. Efficiency of different approaches from the conference version of this paper, in terms of the proportion of ordering of pairs of elements solved, on sets of uniformly distributed random sets of permutations, statistics are generated over 100 000 sets of m = 3,4,5, 10..15, 20 permutations for n=15 and 40 000 sets of m = 3,4,5, 10..15, 20 permutations for n=30.

A first observation comes from the *always theorem* which solves in average, like stated in Section 3, $\frac{1}{2^{m-1}}$ of the ordering of the pairs in an instance of a set of m uniformly generated random permutations. The *always theorem*, which is subsumed in our Major Order Theorems, sets an inferior bound on the efficiency of the approach.

Table 4. I also shows that even if our Major Order Theorem 1.0 is quite stronger than the *always theorem* on small sets of permutations it quickly converge to the *always theorem* as n becomes bigger.

Our Major Order Theorems greatly improve the efficiency on small and medium scale instances (with regards to n). The particular strength of versions 2.0 and 3.0 is to extend the efficiency on big set of permutations (bigger m) where the *always theorem* is hardly applicable.

More complete statistics for the Major Order Theorem 3.0, for bigger m and n can be found in the following Table 4. II.

$m\backslash n$	8	10	15	20	25	30	40	50	60	80	100
3	0.7642	0.7179	0.6345	0.5792	0.5378	0.5059	0.4608	0.4312	0.4086	0.3775	0.3563
4	0.6010	0.5757	0.5201	0.4761	0.4406	0.4127	0.3700	0.3404	0.3176	0.2833	0.2612
5	0.7381	0.6853	0.5813	0.5046	0.4470	0.4038	0.3456	0.3073	0.2793	0.2413	0.2191
10	0.6440	0.6019	0.5173	0.4515	0.4012	0.3617	0.3040	0.2649	0.2354	0.1979	0.1726
15	0.7392	0.6732	0.5445	0.4581	0.3987	0.3544	0.2946	0.2543	0.2254	0.1869	0.1612
20	0.6739	0.6262	0.5248	0.4484	0.3917	0.3486	0.2896	0.2491	0.2209	0.1819	0.1566
25	0.7463	0.6774	0.5440	0.4557	0.3939	0.3497	0.2867	0.2480	0.2186	0.1800	0.1540
30	0.6899	0.6391	0.5307	0.4496	0.3904	0.3464	0.2852	0.2448	0.2164	0.1775	0.1536
35	0.7490	0.6804	0.5467	0.4563	0.3928	0.3474	0.2846	0.2438	0.2156	0.1770	0.1517
40	0.7008	0.6473	0.5349	0.4505	0.3905	0.3448	0.2833	0.2438	0.2142	0.1764	0.1518
45	0.7522	0.6836	0.5484	0.4570	0.3924	0.3473	0.2847	0.2427	0.2141	0.1764	0.1507
50	0.7095	0.6533	0.5367	0.4518	0.3908	0.3450	0.2827	0.2431	0.2138	0.1755	0.1501

TABLEAU 4. II. Efficiency of the Major Order Theorem 3.0, from the conference version of this article, on sets of uniformly distributed random sets of permutations, from $n = 8$ to $n = 100$, $m = 3$ to $m = 50$, statistics are generated over 100 000 sets of permutations for smaller n to 2000 sets of permutations for bigger n.

Theorem 4 greatly improves the efficiency of our method as shown in Table 4. III in comparison to the strict version of Theorem 3 in Table 4. II. Efficiency improvements ranges from +5% (for $n = 100$, $m = 50$) to +46% (for $n = 20$, $m = 4$). The improvement is generally higher for even m, i.e. when the number of permutations in the sets considered is even.

$m\backslash n$	10	15	20	30	40	50	60	80	100	120	150	200	250
3	0.949	0.878	0.808	0.701	0.630	0.578	0.540	0.486	0.450	0.423	0.395	0.365	0.345
4	0.995	0.977	0.937	0.806	0.672	0.573	0.506	0.421	0.369	0.334	0.297	0.259	0.235
5	0.896	0.801	0.716	0.595	0.516	0.461	0.419	0.360	0.319	0.288	0.255	0.218	0.193
10	0.931	0.823	0.709	0.548	0.454	0.393	0.349	0.289	0.250	0.221	0.190	0.156	0.134
15	0.818	0.704	0.612	0.488	0.411	0.358	0.319	0.264	0.227	0.200	0.172	0.140	0.120
20	0.876	0.748	0.637	0.492	0.407	0.351	0.311	0.255	0.217	0.193	0.164	0.133	0.113
25	0.797	0.679	0.587	0.465	0.389	0.337	0.300	0.247	0.211	0.185	0.159	0.129	0.109
30	0.850	0.718	0.610	0.472	0.391	0.337	0.298	0.244	0.208	0.182	0.155	0.125	0.106
35	0.787	0.667	0.575	0.454	0.380	0.328	0.291	0.239	0.204	0.179	0.152	0.123	0.104
40	0.834	0.702	0.596	0.462	0.382	0.330	0.291	0.238	0.203	0.178	0.152	0.122	0.103
45	0.780	0.660	0.568	0.447	0.374	0.323	0.286	0.235	0.200	0.176	0.150	0.121	0.102
50	0.823	0.691	0.587	0.455	0.376	0.324	0.287	0.234	0.200	0.175	0.149	0.1200	0.101
100	0.797	0.665	0.566	0.440	0.365	0.314	0.277	0.227	0.193	0.169	0.143	0.115	0.097

TABLEAU 4. III. Efficiency of the Major Order Theorem 3.0e on sets of uniformly distributed random sets of permutations, from $n = 8$ to $n = 250$, $m = 3$ to $m = 100$, statistics are generated over 1 000 000 sets of permutations for smaller n to 10 000 sets of permutations for bigger n (see Table 4. IV for the exact number of sets for each case).

Our Major Order Theorems exhibit best efficiency on sets of three permutations ($m = 3$), the only case where the theoretical complexity is still not clear. For larger values of n,

n	8, 10, 15, 20, 25, 30	40, 50, 60, 80	100, 120, 150, 200, 250
# sets	1 000 000	100 000	10 000

TABLEAU 4. IV. Number of sets of permutations uniformly generated for each couple $m\backslash n$ (depending only on n)

the Major Order Theorem 3.0 can still solve on average more than 33% of the pairs for 3 random permutations of 100 elements (and more than 45% of the pairs in the MOT3.0e version). Figure 4.1 and Figure 4.2 show the average performance of our approach on 3-permutations and 4-permutations sets of different sizes.

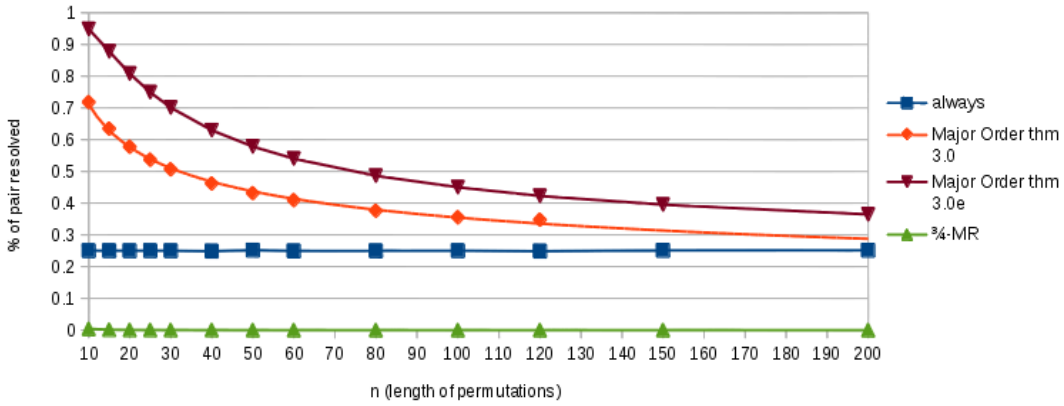


FIGURE 4.1. Efficiency of the Always Theorem, the Major Order Theorems 3.0 and 3.0e and the 3/4 Majority Rule in terms of the proportion of pairs resolution, when $m = 3$ and $n = 10, \ldots, 200$.

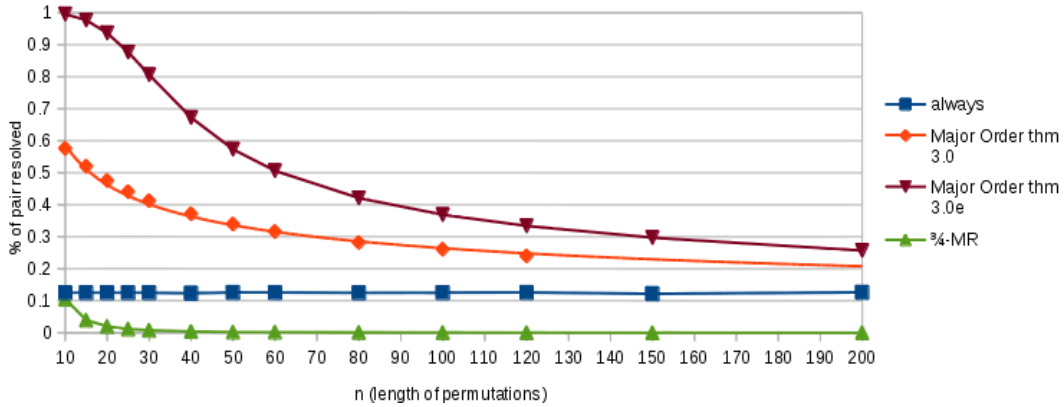


FIGURE 4.2. Efficiency of the Always Theorem, the Major Order Theorems 3.0 and 3.0e and the 3/4 Majority Rule in terms of the proportion of pairs resolution, when $m = 4$ and $n = 10, \ldots, 200$.

To compare our approach with the *3/4 Majority Rule* approach of Betzler *et al.* [18], we first tested its efficiency on uniformly generated sets of m permutations of $[n]$, for different values of m and n. Table 4. V shows that this approach does not perform well on random sets, solving fewer than 1% of the pairs in most of the cases. The case $m = 4$ is an exception, where non-dirty candidates were found in much greater proportions. This lack of efficiency comes from the fact that the *3/4 Majority Rule* is intended for more closely related permutations coming from real life data like competitions (see Section 4.6.2).

We noticed that the vast majority of constraints found by the *3/4 Majority Rule* in instances with sets of uniformly distributed permutations, were also found by the Major Order Theorem 3.0 (see Table 4. VI). Only a few exceptions are found in large scale testing, most of them in instances of the problem with sets of 4 permutations. In approximately 15% of these cases the Major Order Theorem 3.0 does not completely include the *3/4 Majority Rule*.

Our new improved version, Major Order Theorem 3.0 with equality, includes almost all the constraints found by the *3/4 Majority Rule*. Although not theoretically proven, the MOT3.0e version has a statistical inclusion rate (over 1 000 000 instances tested per case) of 100% for $m \neq 4$ (and more than 99% in the case of $m = 4$) meaning that all constraints found by the *3/4 Majority Rule* were also found by the MOT3.0e when $m \neq 4$.

$m \backslash n$	8	10	15	20	30	50	80
3	0.80%	0.41%	0.12%	0.05%	0.01%	0%	0%
4	16.94%	10.48%	3.97%	1.98%	0.78%	0.25%	0.09%
5	2.18%	1.16%	0.37%	0.17%	0.06%	0.01%	ϵ%
6	0.42%	0.20%	0.05%	0.02%	0.01%	ϵ%	0%
10	0.05%	0.02%	ϵ%	ϵ%	0%	0%	0%
15	0%	0%	0%	0%	0%	0%	0%
20	ϵ%	0%	0%	0%	0%	0%	0%

TABLEAU 4. V. Efficiency, in % of resolved pairs, of the *3/4 Majority Rule* on sets of uniformly distributed random permutations, for $n = 8, 10, 15, 20, 30, 50, 80$, $m = 3, 4, 5, 6, 10, 15, 20$, statistics generated over 1 000 000 instances. ϵ, as usual, represent a really small real number > 0.

4.6.2. Efficiency on real data

We compared the efficiency of our Theorem 3 and Theorem 4 to Betzler *et al.* data reduction technique [18] (that includes the 3/4-Majority Rule) on data taken from F1

$m\backslash n$	8	10	15	20	30	50	80
3	100%	100%	100%	100%	100%	100%	100%
4	100%	100%	99.99%	99.96%	99.88%	99.69%	99.68%
5	100%	100%	100%	100%	100%	100%	100%
6	100%	100%	100%	100%	100%	100%	n/a
10	100%	100%	100%	100%	n/a	n/a	n/a
15	n/a	n/a	n/a	n/a	n/a	n/a	n/a
20	100%	n/a	n/a	n/a	n/a	n/a	n/a

TABLEAU 4. VI. Inclusion, in %, of *3/4 Majority Rule* in Major Order Thm 3.0 on the same sets as Table 4. V. We use n/a to represent cases where the *3/4 Majority Rule* was not applicable i.e no dirty candidates were found.

races. Each race (ordering vote) is represented by an ordered list of candidates. For each year (or instance of the problem), we compare the candidates that were present in all races only, like it was done in [18].

Example 2. *For a year that held the following three races :*
$A > B > C > D > E > F > G > H > I$,
$D > E > J > B > A > K$ *and*
$A > D > B > L > E$,
we consider only candidates A, B, D and E, which are present in all races, tagging them with numbers :
$1 > 2 > 0 > 3 > 4 > 0 > 0 > 0 > 0$,
$3 > 4 > 0 > 2 > 1 > 0$ *and*
$1 > 3 > 2 > 0 > 4$.
The resulting set of permutations is $\mathcal{A} = \{1234,3421,1324\}$

We ran our vector implementation of MOT3.0 and MOT3.0e on the sets of F1 races for different years (the data comes from [18] and is also available online here [1]). Table 4. VII shows that MOT3.0 is doing well but that MOT3.0e is doing even better on real data and finds more constraints than the data reduction technique proposed by Betzler *et al.* [18].

As expected, real-world data are generally easier to solve than uniformly random generated data, simply because there is more similarity in real world rankings. As a measure of similarity, the average normalized distances (normalized distance = distance / number of pairs of elements) between permutations of the start set in F1 races were

1. http://www-etud.iro.umontreal.ca/~miloszro/caldam/dam_caldam.html

year	n	m	conflicting pairs	Data reduction from [18]	MOT3.0	MOT3.0e
1975	13	14	100%	64.1%	73.1%	100%
1980	19	14	95.9%	84.2%	77.2%	94.8%
1981	19	15	97.7%	73.1%	83.1%	92.4%
1982	9	16	97.2%	100%	86.1%	100%
1983	24	15	98.9%	38.1%	69.2%	76.1%
1984	19	16	99.4%	94.2%	87.1%	96.5%
1985	14	16	100%	93.4%	84.6%	96.7%
1986	21	16	98.6%	92.9%	84.8%	100%
1987	21	16	99.5%	98.6%	82.9%	99.1%
1988	28	16	94.4%	84.1%	89.4%	98.7%
1989	26	16	88.9%	98.2%	88.6%	99.4%
1990	24	16	90.2%	96.4%	90.9%	96.7%
1991	24	16	94.9%	84.8%	84.4%	90.9%
1992	22	16	99.1%	88.3%	84.9%	100%
1993	18	16	98.7%	91.5 %	83.0%	94.1%
1994	16	16	94.2%	95%	70.8%	100%
1995	16	17	100%	97.5%	98.3%	98.3%
1996	19	16	100%	94.8%	84.8%	100%
1997	18	17	100%	83.0%	91.5%	94.8%
1998	21	16	98.1%	97.2%	91.4%	100%
1999	19	16	97.7%	61.4%	74.3%	84.8%
2000	22	17	99.6%	63.7%	87.0%	88.3%
2001	18	17	99.4%	64.1%	78.4%	82.4%
2002	18	17	91.5%	76.5%	87.6%	92.8%
2003	16	16	98.3%	100%	91.7%	100%
2004	15	18	96.2%	100%	92.4%	100%
2005	13	19	100%	96.2%	96.2%	100%
2006	18	18	99.4%	100%	95.4%	100%
2007	18	17	97.4%	91.5%	93.5%	97.4%
2008	20	18	95.8%	81.1%	90%	94.2%

TABLEAU 4. VII. For each year, "n" defines the length of a permutation (the number of candidates present in all races of the year), "m" defines the number of permutations (number of races in the year), conflicting pairs denotes the percentage of pairs that are not always in the same order in the races of the year, Data reduction from [18] denotes the percentage of pairs solved by Betzler *et al.* data reduction technique, MOT3.0 and MOT3.0e denote the percentage of pairs solved by our MOT3.0 and MOT3.0e approaches.

in the range of 0.31 to 0.44 as the average normalized distances of the start set in randomly generated data of the same size where 0.47. Following Mallow's model [89], the permutations had a dispersion parameter θ ranging from 0.137 to 0.357. Mallow's model can be written as :

$$P_{\pi_0,\theta}(\pi) = \frac{e^{-\theta d_{KT}(\pi,\pi_0)}}{\psi(\theta)}$$

Where $\psi(\theta) = \sum_{\pi \epsilon S_n} e^{-\theta d_{KT}(\pi,\pi_0)}$. Randomly generated data have a dispersion parameter θ of 0. As θ grows, the permutations of $\mathcal{A}$ become closer to π_0, the "center" of dispersion of this model.

4.6.3. Time complexity of our approach and implementation

Our older implementation of the Major Order Theorem 3.0 (in [96]) had a theoretical complexity of $O(n^3mk)$, where n is the size of the permutations, m is the number of permutations in $\mathcal{A}$ and k is the number of iterations of the last refined version of our approach (Theorem 4).

The new vector version of the Major Order Theorem 3.0 with equality (Theorem 4') has an initial complexity of $O(n^2m + n^3 + n^2k)$. Constructing the right matrix R (or the left matrix L) takes $O(n^2m)$ and constructing the difference vectors $\overrightarrow{R_i} - \overrightarrow{R_j}$ and Δ_{ij} takes $O(n^3)$. For each of the $\frac{n(n-1)}{2}$ pair of elements, we check if the theorem is applicable in $O(1)$. We repeat the process over all the pairs k times, until no more new constraints are found, giving us an additional complexity of $O(n^2k)$. As we can see in Table 4. VIII, the average number of iterations k is between 4 and 7 (The biggest recorded number of iterations was $k = 25$ in a case where $n = 50$ and $m = 4$, on more than 96 000 000 random instances). Every time we find a new constraint, we iterate twice over all other elements, to check if we can find a new constraint by transitivity and we iterate twice over all other elements to update the difference vectors $\overrightarrow{R_i} - \overrightarrow{R_j}$, adding an additional $O(n)$ to the complexity. Since there are $\frac{n(n-1)}{2}$ possible pairs, there are no more than $\frac{n(n-1)}{2}$ transitivity scans and difference vectors updates thus resulting in an amortized complexity of $O(n^3)$. As $k < n$, the complexity of our approach is $O(n^2m + n^3)$.

As an example, on an Intel(R) Core(TM) i7 CPU 870 @ 2.93GHz, computing the Major Order Theorem 3.0e constraints for an instance with $m = 3$ and $n = 400$ take less than 2 seconds, where it was less than 30 seconds with the non vectorial version.

Our implementation has a $O(n^3)$ space complexity. In practice, our code runs out of memory for sets of permutations of length 750 but still works on permutations of length 700. Our biggest computed set had $m = 1000$ permutations of $n = 700$ elements and took 20 seconds to preprocess, solving 9632/244650 (3.94%) of the pairs.

A simple Branch and Bound solver, using some basic left/right constraints from [26] and cutting non-promising branches, was previously combined with an implementation of the Major Order Theorem 3.0 in order to evaluate the gain of using the MOT constraints

$m\backslash n$	10	15	20	30	40	50	60	80	100	120	150	200	250
3	2.88	3.339	3.594	3.878	4.057	4.182	4.278	4.433	4.577	4.709	4.89	5.141	5.335
4	3.221	4.191	5.221	6.787	7.351	7.417	7.378	7.265	7.148	7.084	6.977	6.808	6.687
5	3.092	3.550	3.835	4.179	4.389	4.551	4.692	4.918	5.102	5.218	5.377	5.601	5.745
10	3.532	4.389	4.782	4.978	5.037	5.077	5.115	5.180	5.235	5.297	5.346	5.401	5.463
15	3.210	3.671	3.946	4.244	4.415	4.541	4.627	4.772	4.874	4.948	5.035	5.145	5.250
20	3.533	4.174	4.443	4.635	4.732	4.797	4.859	4.947	4.999	5.061	5.135	5.210	5.252
25	3.238	3.678	3.936	4.217	4.379	4.487	4.575	4.706	4.808	4.855	4.951	5.047	5.134
30	3.511	4.076	4.318	4.519	4.630	4.701	4.767	4.857	4.922	4.982	5.043	5.110	5.166
35	3.250	3.677	3.927	4.200	4.359	4.467	4.553	4.673	4.760	4.830	4.920	5.011	5.068
40	3.490	4.016	4.251	4.457	4.572	4.652	4.711	4.808	4.867	4.932	4.992	5.0550	5.127
45	3.255	3.677	3.922	4.189	4.344	4.451	4.532	4.653	4.745	4.817	4.880	4.968	5.034
50	3.474	3.975	4.206	4.415	4.533	4.614	4.683	4.773	4.848	4.908	4.950	5.030	5.090
100	3.425	3.877	4.098	4.319	4.441	4.530	4.601	4.699	4.770	4.816	4.889	4.958	5.021

TABLEAU 4. VIII. Average number of iterations of the Major Order Theorem 3.0e on sets of uniformly distributed random sets of permutations, from $n = 8$ to $n = 250$, $m = 3$ to $m = 100$, statistics generated over 1 000 000 sets of permutations for smaller n to 10 000 sets of permutations for bigger n (see Table 4. IV for the exact number of sets for each case).

in solving the median problem. The original Branch and Bound solver took 13 minutes and 53 seconds to solve 1000 uniformly random instances of $m = 3$ and $n = 15$. When combined with our MOT3.0, the calculation time was reduced to a mere 10 seconds, and with the MOT3.0e vector version, to 1.5 seconds. The source code (Java, using standard libraries) and real data are available [2] for testing and replicating the experimental results.

4.7. Conclusion and future works

In this paper a new approach was presented that partly solve the median problem, under the Kendall-τ distance, by finding a set of ordering constraints on pairs of elements in a median. The approach's scope is larger than previous approaches (*always theorem, 3/4 Majority Rule*). Therefore, it is much more efficient on data, especially uniformly-distributed data, which is well-reflected on showed statistics. Our approach has a great efficiency on small and medium scale instances of the problem and, curiously, has an even greater impact on cases where $m = 3$. The constraints found by this approach may be used in any algorithm or heuristic to accelerate computations.

Further works include the search for more constraints based on pairwise properties of elements given their positions in the permutations of $\mathcal{A}$. Is the proximity of positions for a pair of elements a clue for their relative positions in the median ? We are also interested in solving the problem exactly using a combination of constraints, data reduction, strong

2. http://www-etud.iro.umontreal.ca/~miloszro/caldam/dam_caldam.html

approximation and optimization techniques. Preliminary works show that we can find the exact median distance for randomly uniformly generated data of $m = 3$ permutations of size $n = 50$ in less than 2 seconds, on average, using a Branch-and-Bound algorithm.

The greater efficiency of the Major Order Theorems for the special case where $m = 3$ shows great promise for further work. Are there other additional properties, in this particular case, which may further enhance the efficiency ? The complexity remains unknown for $m = 3$ and even if it looks easier to solve, it would be interesting to find its theoretical complexity.

Acknowledgements. Thanks to Bryan Brancotte, Sarah Cohen-Boulakia and Alain Denise (LRI - Paris Sud) for giving us useful advice and thoughts to guide the work. This work was supported by a grant from the National Sciences and Engineering Research Council (NSERC) through an Individual Discovery Grant 262965-11 (Hamel) and by Fonds de recherche Nature et Technologie du Québec (FRQNT) through a Master's scholarship (Milosz).

Chapitre 5

HEURISTIQUES, ALGORITHMES ET RÉDUCTION D'ESPACE

Résumé

L'article "Heuristic, Branch-and-Bound Solver and Improved Space Reduction for the Median of Permutations Problem" a été présenté à la conférence International Workshop on Combinatorial Algorithms (IWOCA), en juillet 2017 à Newcastle en Australie [97].

L'article présente trois approches différentes pour résoudre le problème de la médiane de permutations. La première approche est celle de l'heuristique du recuit simulé ("Simulated Annealing" ou "SA") qui a été bien paramétré à l'aide de nombreux tests. L'heuristique donne une bonne approximation qui est souvent la solution optimale (dans plus de 96% des cas) pour les tailles d'instances étudiées.

La seconde approche combine la réduction d'espace de recherche par les Major Order Theorems [96] avec la borne inférieure de Conitzer et al. [44] pour donner une borne inférieure plus stricte. Lorsque combiné à l'approximation par recuit simulé, cette borne inférieure stricte augmente la réduction de l'espace de recherche en ajoutant des contraintes supplémentaires sur la forme de la solution optimale. Cette méthode est nommée "Lower Upper Bound Constraints" (LUBC). L'efficacité de l'approche est démontrée avec des statistiques basées sur des instances aléatoires.

La troisième et dernière approche est la résolution exacte du problème par un algorithme branch-and-bound qui construit les permutations médianes. L'algorithme utilise les deux approches précédentes pour guider l'exploration et couper les sous-espaces de recherche qui ne contiennent pas de solutions optimales.

Une version journal de cet article, contenant la combinaison des deux premières approches avec une formulation en nombres entiers du problème et CPLEX a été acceptée au Journal of Discrete Algorithms. Cette version journal est brièvement résumée à la section 5.8 et peut être trouvée à l'annexe B.

Partage du travail

Dans les contributions de cet article, Robin Milosz a implémenté, testé et paramétré l'heuristique du recuit simulé. Robin Milosz et Sylvie Hamel ont développé et prouvé la réduction d'espace de la seconde approche. Robin Milosz a généré les statistiques pour mesurer l'efficacité de cette approche. Robin Milosz a développé et implémenté l'algorithme exact du branch-and-bound. Les implémentations sont en Java et disponibles sur http://www-etud.iro.umontreal.ca/~miloszro/iwoca/iwoca.html. Tous les auteurs ont rédigé l'article.

Heuristic, Branch-and-Bound Solver and Improved Space Reduction for the Median of Permutations Problem

Robin Milosz and Sylvie Hamel

DIRO - Université de Montréal ,
C. P. 6128 Succ. Centre-Ville, Montréal, Qc, Canada, H3C 3J7

5.1. Abstract

Given a set $\mathcal{A} \subseteq \mathbb{S}_n$ of m permutations of $\{1,2,\ldots,n\}$ and a distance function d, the **median** problem consists of finding a permutation π^* that is the "closest" of the m given permutations. Here, we study the problem under the Kendall-τ distance which counts the number of order disagreements between pairs of elements of permutations. In this article, we explore this NP-hard problem using three different approaches : a well parameterized heuristic, an improved space search reduction technique and a refined branch-and-bound solver.

5.2. Introduction

The problem of finding medians of a set of m permutations of $\{1,2,\ldots,n\}$ under the Kendall-τ distance [**78**], often cited as the Kemeny Score Problem [**76**] consists of finding a permutation that agrees the most with the order of the m given permutations, *i.e.*, that minimizes the sum of order disagreements between pairs of elements of permutations. This problem has been proved to be NP-hard when $m \geq 4$, m even (first proved in [**56**], then corrected in [**23**]), but its complexity remains unknown for $m \geq 3$, m odd. A lot of work as been done in the last 15 years, either on deriving approximation algorithms [**3, 80, 135**] or fixed-parameter ones [**22, 75, 104**]. Other theoretical approaches aiming at reducing the search space for this problem have also been developed [**18, 26, 44, 51, 98**].

In this present work, we are interested in solving methods for the median of permutations problem, focusing on three different approaches. After introducing some basic definitions and notations in Section 5.3, we present our first approach in Section 5.4, an adaptation of the well known "Simulated Annealing" heuristic to our context. Second, in Section 5.5, we built ordering constraints for pairs of elements appearing in a median

by merging previous approches [44, 98] complemented by our simulated annealing heuristic, thus reducing significantly the search space for this median. Third, we present, in Section 5.6, an implementation of an exact solver : a branch-and-bound algorithm that is powered by the two previous approaches. Finally, Section 5.7 gives some thoughts on future works.

5.3. Median of permutation : definitions and notations

A **permutation** π is a bijection of $[n] = \{1,2\ldots,n\}$ onto itself. The set of all permutations of $[n]$ is denoted $\mathbb{S}_n$. As usual we denote a permutation π of $[n]$ as $\pi = \pi_1\pi_2\ldots\pi_n$, and a segment of π by $\pi[i..j] = \pi_i\pi_{i+1}...\pi_{j-1}\pi_j$. The cardinality of a set $\mathcal{S}$ will be denoted $\#\mathcal{S}$.

The **Kendall-τ distance**, denoted d_{KT}, counts the number of order disagreements between pairs of elements of two permutations and can be defined formally as follows : for permutations π and σ of $[n]$, we have that

$$d_{KT}(\pi,\sigma) = \#\{(i,j)|i<j \text{ and } [(\pi[i]<\pi[j] \text{ and } \sigma[i]>\sigma[j]) \text{ or } (\pi[i]>\pi[j] \text{ and } \sigma[i]<\sigma[j])]\},$$

where $\pi[i]$ denotes the position of integer i in permutation π.

Given any set of permutations $\mathcal{A} \subseteq \mathbb{S}_n$ and a permutation $\pi \in \mathbb{S}_n$, we have $d_{KT}(\pi,\mathcal{A}) = \sum_{\sigma\in\mathcal{A}} d_{KT}(\pi,\sigma)$. The **problem of finding a median of $\mathcal{A}$ under the Kendall-τ distance** can be stated formally as follows : Given $\mathcal{A} \subseteq \mathbb{S}_n$, we want to find a permutation π^* of $\mathbb{S}_n$ such that $d_{KT}(\pi^*,\mathcal{A}) \leq d_{KT}(\pi,\mathcal{A})$, $\forall\ \pi \in \mathbb{S}_n$. Note that a set $\mathcal{A}$ can have more than one median. To keep track of the number of permutations in $\mathcal{A}$ that have a certain order between two elements, let us introduce the left/right distance matrices L and R.

Definition 5. *Let $L(\mathcal{A})$, be the* **left distance matrix** *of a set of m permutations $\mathcal{A} \subseteq \mathbb{S}_n$, where $L_{ij}(\mathcal{A})$ denotes the number of permutations of $\mathcal{A}$ having element i to the left of element j. Symmetrically, let $R(\mathcal{A})$, be the* **right distance matrix** *of $\mathcal{A}$, where $R_{ij}(\mathcal{A})$ denotes the number of permutations of $\mathcal{A}$ having element i to the right of element j. Obviously, $L_{ij}(\mathcal{A}) + R_{ij}(\mathcal{A}) = m$ and $L_{ij}(\mathcal{A}) = R_{ji}(\mathcal{A})$.*

We can calculate the distance between a permutation $\pi \in \mathbb{S}_n$ and a set of permutations $\mathcal{A} \subseteq \mathbb{S}_n$ using the right (or left) distance matrix as follow :

$$d_{KT}(\pi,\mathcal{A}) = \sum_{i=1}^{n} \sum_{\substack{j=1,j\neq i \\ \pi[j]>\pi[i]}}^{n} R_{ij}(\mathcal{A}) = \sum_{i=1}^{n} \sum_{\substack{j=1,j\neq i \\ \pi[j]>\pi[i]}}^{n} L_{ji}(\mathcal{A}).$$

5.4. A heuristic approach

Our first approach is based on the well known Simulated Annealing (SA) heuristic. This approach will give us an approximative solution for the median problem, an upper bound on the distance we are trying to minimize ($d_{KT}(\pi, \mathcal{A})$) and a direction for our branch-and-bound search.

5.4.1. Simulated Annealing

Simulated annealing (SA) is a probabilistic metaheuristic for locating a good approximation to the global optimum of a given function in a large search space [**113**]. It is an adaptation of the Metropolis-Hasting algorithm [**93**] and works best on large discrete space. For that reason, it is a good choice in our case where the search space is $\mathbb{S}_n$, the space of all permutations of $[n]$.

In a simulated annealing heuristic, each point s of the search space corresponds to a state of a physical system, and minimizing a function $f(s)$ corresponds to minimizing the internal energy of the system in state s. The goal is to bring the system, from a randomly chosen initial state, to a state with the minimum possible energy. At each step, the SA heuristic considers a neighbour s' of the current point s and (1) always moves to it if $f(s') < f(s)$ or (2) moves to it with a certain probability if $f(s') > f(s)$. This probability to do a "bad" move is almost 1 at the beginning of the heuristic when the system is heated but goes to 0 as the system is cooled. The heuristic stops either when the system reaches a good enough solution or when a certain number of moves has been made.

In our context, we are given a set of permutations $\mathcal{A} \subseteq \mathbb{S}_n$ for which we want to find a median. The function f we need to minimize, given the set $\mathcal{A}$, is $f(\pi) = d_{KT}(\pi,\mathcal{A})$, for $\pi \in \mathbb{S}_n$. We first choose randomly a starting point in our space i.e. a permutation $\pi \in \mathbb{S}_n$. This permutation is uniformly generated using the Fisher-Yates shuffle [**64**]. To find neighbours of a permutation π in our system, we consider circular moves defined as follows :

Definition 6. *Given $\pi \in \mathbb{S}_n$, we call* **circular move** *of a segment $\pi[i..j]$ of π, denoted $c[i,j](\pi)$, the cycling shifting of one position to the right, if $i < j$, of this segment inside the permutation π : $c[i,j](\pi) = \pi_1 \dots \pi_{i-1}\pi_{\mathbf{j}}\pi_{\mathbf{i}} \dots \pi_{\mathbf{j-1}}\pi_{j+1} \dots \pi_n$ (if $i > j$, we shift to the left : $c[j,i](\pi) = \pi_1 \dots \pi_{j-1}\pi_{\mathbf{j+1}} \dots \pi_{\mathbf{i}}\pi_{\mathbf{j}}\pi_{i+1} \dots \pi_n$).*

So, to choose a neighbour for our current permutation π, our SA heuristic first randomly generates two integers $i \neq j$, $1 \leq i, j \leq n$, and compute neighbour(π)= $c[i,j](\pi)$, if $i < j$ or neighbour(π)=$c[j,i](\pi)$, otherwise. To decide whether or not we move into state

neighbour(π), we compute the difference of energy, noted ΔE, which in our case is the difference $d_{KT}(neighbour(\pi)),\mathcal{A}) - d_{KT}(\pi,\mathcal{A})$. If this difference is negative or null, the state neighbour(π) is closer to the median of $\mathcal{A}$ and we move to this state. If it is positive, we move to state neighbour(π) depending on the acceptance probability $e^{-\Delta E/T}$, where T is the temperature of the system. This process of going to neighbours states is repeated a fixed number of times during which the permutation with the lowest energy (distance to $\mathcal{A}$) are kept in a set. This set is returned at the end of the process. Algorithm 1, in Appendix A.I, gives the pseudo-code of our heuristic SA [1].

5.4.2. Choosing the parameters

The important parameters of a SA heuristic are : the initial temperature of the system, the cooling factor, the maximal number of movements, the function that propose a neighbouring alternative and the number of repetitions of SA. First, we choose the circular move (see Definition 6) as our neighbouring choosing function. This choice was made for two reasons. First, while looking at sets of medians for different sets of permutations $\mathcal{A} \subseteq \mathbb{S}_n$, we observe that most of the medians in a set could be obtained from one another by applying those circular moves. Second, we did try a lot of other moves (exchanging element $\pi[i]$ with one of its neighbour $\pi[i-1]$ or $\pi[i+1]$ or exchanging it with any other element $\pi[j]$, inverting the order of the elements of a block, etc.) that clearly did not converge as quickly as the circular moves.

For the rest of the SA parameters, note that for each instance of our problem, i.e. a set of permutations $\mathcal{A} \subseteq \mathbb{S}_n$, we have two important parameters : the number m of permutations in $\mathcal{A}$ and the size n of the permutations. This pair (m,n) is very relevant to describe the problem's difficulty. So, we were interested in tuning the SA parameters as function of n and m.

We made extensive testing to find out the optimal parameters for SA given the pair (m,n). More information on our choice of parameters are available in Appendix A. II. But here is an overview :

1. Initial solution : a random permutation generated by Fisher-Yates shuffle.
2. The cooling schedule is : $t_i = \alpha t_{i-1}$.
3. The initial temperature of the system set to : $t_0 \longleftarrow (0.25m + 4.0)n$

1. Source code (Java) for testing can be found online at
http ://www-etud.iro.umontreal.ca/~miloszro/iwoca/iwoca.html

4. The cooling factor set to :

$$\alpha = \begin{cases} 0.99 & \text{if } m = 3,4 \\ 0.95 & \text{if } n \leq 10 \\ 0.99 & \text{if } 11 \leq n \leq 16 \\ 0.999 & \text{if } 17 \leq n \leq 20 \\ 0.9995 & \text{if } 21 \leq n \leq 24 \\ 0.9998 & \text{otherwise} \end{cases}$$

5. The neighbour generating function : the circular move.

6. The number of allowed movements for a solution :

$$nbMvts = \begin{cases} 0.6n^3 - 11n^2 + 127n & \text{if } m = 3 \\ 0.9n^3 - 29n^2 + 435n - 1623 & \text{if } m = 4 \\ 250 & \text{if } n \leq 7 \\ 90n^2 - 1540n + 7000 & \text{if } 8 \leq n \leq 24 \\ 35n^2 - 660n + 31000 & \text{if } 25 \leq n \leq 38 \\ 80n^2 - 2300n + 27000 & \text{if } n > 38 \end{cases}$$

7. The number of times to repeat the SA heuristic :

$$nbRuns = \begin{cases} \lceil 0.05n + 2 \rceil & \text{if } m = 3,4 \\ \lceil 0.007nm + 3 \rceil & \text{if } m \text{ is odd} \\ \lceil 0.002mn + 3 \rceil & \text{if } m \text{ is even} \end{cases}$$

The initial temperature was set to accept (almost) all neighbours in the first iterations. The cooling constant and the number of movements where chosen to have a good compromise between the success probability (probability of achieving the optimal score) and the computing time for each (m,n). The number of SA runs was set such that the probability of overall success is above 96% for all instances $n \leq 38$ and $m \leq 50$. For greater m and n, we extrapolated the data to predict a similar SA behaviour.

As $m = 3,4$ are unique cases, that do not seem to be affected by the cooling constant in our considered range and seem much easier to optimize, we treated them appart. For the rest, we found out that m odd problems are harder to solve than m even problems,

and required more runs to reach the same success rate. Without surprise, as n and m are getting bigger, the problem is harder to solve.

5.5. Space reduction technique

In [96] and [98] we found theoretical properties of a set of permutations $\mathcal{A} \subseteq \mathbb{S}_n$ (called the **Major Order Theorems**) that can solve the relative order between some pairs of elements in median permutations of $\mathcal{A}$ thus reducing the search space. In this section we will show how to find additional constraints for the problem by merging this previous work on constraints with a lower bound idea of Conitzer *et al.* ([44]), giving us an even stronger lower bound, that can be then combined with the upper bound obtained by the simulated annealing method presented in Section 5.4. But first, let us quickly recalled our major order theorems in Section 5.5.1 and Conitzer *et al.* lower bound idea in Section 5.5.2.

5.5.1. Some constraints

Given a set of permutations $\mathcal{A} \subseteq \mathbb{S}_n$, the major order theorems, presented in [96], solve the relative order between some pairs of elements in median permutations of $\mathcal{A}$. Thus, these theorems build a set of constraints that can be represented as a boolean matrix C, where $C_{ij} = 1$ if and only if we know that element i will precede element j (denoted $i \prec j$) in median permutations of $\mathcal{A}$. Note that this set of constraints reduces the search space for a median by cutting off all permutations breaking at least one constraint.

Here, we resume the ideas behind these theorems [2] but first, let us formally define the major order between pairs of elements.

Definition 7. *Let $\mathcal{A} \subseteq \mathbb{S}_n$ be a set of permutations. Let $L(\mathcal{A})$ and $R(\mathcal{A})$ be the left and right matrices of $\mathcal{A}$ as defined in Definition 5. Given two elements i and j, $1 \leq i < j \leq n$, we say that the* **major order** *between elements i and j is $i \prec j$ (resp. $j \prec i$) if $L_{ij}(\mathcal{A}) > R_{ij}(\mathcal{A})$ (resp. $R_{ij}(\mathcal{A}) > L_{ij}(\mathcal{A})$), the* **minor order** *is then $j \prec i$ (resp. $i \prec j$). We use δ_{ij} to denote the absolute value of the difference between the major and minor order of two elements i and j.*

The idea of the first major order theorem (**MOT1**) relies on the proximity of two elements in permutations of $\mathcal{A}$: if $i \prec j$ is the major order of elements i and j in permutations of $\mathcal{A}$, we can say that i will also be placed before j in all medians of $\mathcal{A}$ if the numbers of elements (including possible copies) between i and j in those permutations of $\mathcal{A}$ where

2. More detailed explanations and some examples for these Major Order Theorems can be found in Section 4 of [96].

$j \prec i$ is **less than** δ_{ij}. Intuitively, this multiset of elements between i and j, that we will called **interference multiset**, act as interference to the major order $i \prec j$ and so, if its cardinality is small enough, MOT1 gives a relative order for the pair (i,j) in medians of $\mathcal{A}$.

The second major order theorem (**MOT2**) built on the first one by reducing the cardinality of the interference multiset of a pair of elements i and j by removing from it any element that also appears in between i and j in permutations of $\mathcal{A}$ that follows the major order of this pair.

The idea of the third Major Order Theorem (**MOT3**) is to use previously found constraints (MOT1 and MOT2) to reduce even more the cardinality of the interference multiset of a pair of elements i and j, by removing from it all elements that cannot be in it. As an example, say that an element k is in the interference multiset of pair (i,j). This means that k appears in between i and j in at least one permutation of $\mathcal{A}$ where i and j are in their minor order. If we have already found using MOT1 or MOT2 that $C_{ki} = C_{kj} = 1$ or that $C_{ik} = C_{jk} = 1$ then k cannot be in between i and j and we can remove it from the interference multiset. This process is repeated until no new constraint is found.

Example 3. *The MOT3 is better illustrated with the following exemple : for* $\mathcal{A} = \{78\underline{2}36\underline{1}54, 35\underline{1}786\underline{2}4, 5834\underline{1}\underline{2}76\}$*, the major order for* 1 *and* 2 *is* $1 \prec 2$*,* $\delta_{12} = 1$ *and we have* {3,6} *as the interference multiset. The* 6 *gets cancelled by the* 6 *between* 1 *and* 2 *in the second permutation (MOT2's way) as the* 3 *is eliminated by contraints* $3 \prec 1$ *and* $3 \prec 2$ *which were found by MOT1. Therefore the interference multiset is empty and* $1 \prec 2$ *is a valid constraint.*

In [98], we extended those major order theorems by considering the equality case (denoted MOTe) *i.e* the extended MOT theorems gives us the relative order of a pair of elements if the cardinality of the interference multiset for this pair is **less than or equal to** δ_{ij}. This case is more delicate as the method builds constraints only for a subset of the set of median permutations of $\mathcal{A}$, so care is to be taken to avoid possible contradicting constraints (it is possible that in one median of $\mathcal{A}$, $i \prec j$ and in another $j \prec i$; so using the MOTe theorems we will strictly find those medians of $\mathcal{A}$ satisfying one and only one of those “contradicting" constraint). However, the MOTe theorems have a much better efficiency than the MOT theorems, as you can see in Table 5. I.

5.5.2. A new lower bound

Given a set of permutations $\mathcal{A} \subseteq \mathbb{S}_n$, Davenport and Kalagnanam [51] propose a first intuitive lower bound on the median problem of $\mathcal{A}$: the sum, for all pairs of elements i, j, $1 \leq i < j \leq n$, of the number of permutations in $\mathcal{A}$ having the pair (i,j) in its minor order. This bound corresponds to the possibility of ordering all pairs in a median of $\mathcal{A}$ with respect to their major order. If this ordering is possible without conflicting pairs, the bound is equal to the Kendall-τ distance of a median to $\mathcal{A}$. The bound can be easily compute by the following formula :

$$LowerBound_0 = \sum_{\substack{i<j \\ i,j\in[n]}} min\{R_{ij},R_{ji}\}$$

Consider the directed weighted graph $G = (V,E)$, where each vertices $v \in V$ is a element of $[n]$ and where E is composed of two edges for each pair of vertices i,j : e_{ij} and e_{ji} with respected weights $w(e_{ij}) = R_{ji}(\mathcal{A})$ and $w(e_{ji}) = R_{ij}(\mathcal{A})$. The median problem can be reformulated as a minimum feedback arc set problem [74] which consist of finding the set of edges $E^* \subset E$ of minimal total weight $w(E^*) = \sum_{e\in E^*} w(e)$ to be removed from G to obtain a direct acyclic graph. Let G' be the graph obtained from G by "cancelling", for each pair of vertices, opposing ordering in permutations of $\mathcal{A}$: $w(e'_{ij}) = w(e_{ij}) - min\{R_{ij},R_{ji}\}$, $\forall 1 \leq i < j \leq n$. Obviously $w(E^*) = w(E'^*) + LowerBound_0$.

Let $DC_{G'}$ be a set of disjoint directed cycles of G'. The previous lower bound can be augmented by adding for each cycle $c \in DC_{G'}$ the minimal weight of one of its edges :

$$LowerBound_1 = LowerBound_0 + \sum_{c\in DC_{G'}} \min_{e\in c}\{w(e)\}.$$

In [44], Conitzer, Davenport and Kalagnanam push further the lower bound, proving that the cycles do not need to be disjoint. Let $JC_{G'}$ be any sequence $c_1, c_2, ..., c_l$ of G', that can share commun edges. Let $I(c,e)$ be an indicator function of the inclusion of an edge e to a cycle c, *i.e.* $I(c,e) = 1$ if $e \in c$ and 0 otherwise. If $v_i = \min_{e\in c_i}\{w(e) - \sum_{j=1}^{i-1} I(c_j,e)v_j\}$, then we obtain the new following lower bound :

$$LowerBound_2 = LowerBound_0 + \sum_{i=1}^{l} v_i.$$

In practice, we can apply this lower bound method by iteratively searching for a cycle c in G', containing only non-zero weight edges, finding its minimal weight edge e, then

subtracting $w(e)$ to the weight of all edges of c and adding it to the lower bound. This process is then repeating until no such cycle is left in G'.

The process of finding the strongest lower bound, *i.e.* the best sequence and choice of cycles, becomes a problem itself and can be resolved using linear programming. As we are interested here by an efficient pre-processing of the problem, we will use a restrained version of this previous lower bound that can be calculated quickly. Thus, only cycles of length 3 (3-cycles) will be considered.

Our contribution resides in taking advantage of a set of constraints (the one described in Section 5.5.1) while calculating $LowerBound_2$ as it provides additional information on the structure of the optimal solution. If $C_{ij} = 1$ then we know that the order of elements i and j in a median permutation of $\mathcal{A}$ will be $i \prec j$. In that case, we can add $w(e_{ji})$ to the lower bound (since all permutations with $j \prec i$ in $\mathcal{A}$ disagree with a median permutation) then set its value to $w(e_{ji}) = 0$ in G'.

At first glance, incorporating the constraints seems to be interesting but one will quickly observe that the constraints previously found by the MOT method are only of the type $C_{ij} = 1$ where $i \prec j$ is the major order, adding nothing to the lower bound because in the graph G', the associated minor order edge e'_{ji} has weight $w(e'_{ji}) = 0$, by construction.

Nevertheless, the superiority of calculating a lower bound with constraints will appear in Section 5.5.4, when combined with an upper bound.

For $\mathcal{A} \subseteq \mathbb{S}_n$, we will denote the lower bound with set of constraints C by $Lb_{\mathcal{A}}(C)$. If $C = \varnothing$, then $Lb_{\mathcal{A}}(\varnothing)$ becomes $LowerBound_2$ associated with 3-cycles, described above.

5.5.3. An upper bound

In Section 5.4, we detailed a simulated annealing heuristic that finds a approximative solution for our problem. The approximative solution is a valid permutation therefore its distance to $\mathcal{A}$ will be used as an upper bound. We will denote this upper bound by $Ub_{\mathcal{A}}$. Naturally, if $Lb_{\mathcal{A}}(C) = Ub_{\mathcal{A}}$ for a particular instance of the problem and any set of valid constraints C, then the problem is solved : the median distance is $Ub_{\mathcal{A}}$ and the associated permutation to that upper bound will be a solution *i.e.* a median of $\mathcal{A}$.

5.5.4. Putting everything together

Given a set of permutations $\mathcal{A} \subseteq \mathbb{S}_n$, we can deduce a valid set of constraints C using the MOT methods described in Section 5.5.1 and then apply the technique described in

Section 5.5.2 to obtain the lower bound $Lb_{\mathcal{A}}(C)$. Running the SA heuristic of Section 5.4 will get us the upper bound $Ub_{\mathcal{A}}$.

We can use these upper and lower bounds to search for new constraints simply by adding a new possible constraint and verifying if the lower bound has exceeded the upper bound with this new add on. To do so, let us choose a pair of elements (i,j) for which the ordering is still unknown in the median, *i.e.* for which $C_{ij} = 0$ and $C_{ji} = 0$. Now, let us suppose that $i \prec j$ in a median of $\mathcal{A}$ and let C' be the set of constraints C augmented with this new constraint $C_{ij} = 1$. If $Lb_{\mathcal{A}}(C') > Ub_{\mathcal{A}}$ then the added constraint $i \prec j$ is false, which means that $j \prec i$ in a median of $\mathcal{A}$ and we can add $C_{ji} = 1$ to our set of constraints C.

As finding a new constraint give an advantage to find others (as there is more knowledge about the structure of the optimal solution), we can redo the same process including unknown constraints that previously failed the test, repeating until no new constraint is found. We will called this new way of finding constraints the LUBC (lower-upper bounds and constraints) method.

Without surprise, as MOT3 method also benefits from new constraints, we propose a method that will iteratively alternate between MOT and LUBC until both are unable to find any additional constraint and call it MOT+LUBC. In Appendix B.I, Algorithm 2 and Algorithm 3 give the pseudo-code of our MOT+LUBC method.

As seen before, the constraints found by the MOT method are only of the type $C_{ij} = 1$ where $i \prec j$ is the major order. An advantage of the LUBC is the possibility to find constraints of $\mathcal{A}$ that are of the minor type i.e. where $C_{ij} = 1$ and $i \prec j$ is the minor order $(R_{ij} > R_{ji})$. Recalling the construction of G', the weight of an edge associated with the major order is strictly positive and leads to a non-zero augmentation of the lower bound. When the LUBC methods finds any new valid constraint of the minor order type, the augmentation of the lower bound is advantageous to find further new contraints.

The great efficiency of this new method can be observed in Table 5. I, where we tested our different approaches on distributed random sets of m permutations of $[n]$, with $m \in [3; 50]$ and n = 15,20,30 or 45. On our instances, the gain of constraints is ranging from +1% to 41% passing from MOT to MOT+LUBC and from +0.1% to 36% passing from MOTe to MOTe+LUBC. We can note that all instances of $n \leq 20$ have a average resolution rate higher than 90% with the MOTe+LUBC method.
Tables 10 to 14 in Appendix B.II, gives more results for all of these methods. Appendix B.III discuss the time complexity of this new approach.

	$n = 15$				$n = 20$				$n = 30$				$n = 45$			
m	MOT	MOT+ LUBC	MOTe	MOTe+ LUBC	MOT	MOT+ LUBC	MOTe	MOTe+ LUBC	MOT	MOT+ LUBC	MOTe	MOTe+ LUBC	MOT	MOT+ LUBC	MOTe	MOTe+ LUBC
3	0.635	0.921	0.878	0.966	0.579	0.885	0.808	0.952	0.506	0.725	0.702	0.873	0.447	0.513	0.604	0.669
5	0.595	0.913	0.800	0.954	0.530	0.859	0.715	0.929	0.444	0.605	0.595	0.766	0.361	0.381	0.490	0.516
10	0.524	0.845	0.824	0.951	0.465	0.809	0.709	0.915	0.384	0.616	0.549	0.747	0.310	0.347	0.419	0.454
15	0.579	0.939	0.704	0.953	0.500	0.890	0.612	0.919	0.400	0.569	0.490	0.668	0.316	0.328	0.383	0.398
20	0.540	0.884	0.747	0.945	0.472	0.843	0.636	0.908	0.383	0.597	0.493	0.697	0.306	0.327	0.377	0.399
25	0.581	0.949	0.680	0.923	0.500	0.903	0.587	0.923	0.398	0.579	0.465	0.653	0.312	0.322	0.363	0.375
30	0.550	0.904	0.720	0.947	0.479	0.861	0.610	0.910	0.386	0.589	0.472	0.674	0.301	0.317	0.362	0.379
35	0.584	0.955	0.668	0.961	0.501	0.909	0.575	0.926	0.396	0.582	0.453	0.642	0.309	0.320	0.345	0.356
40	0.556	0.915	0.702	0.950	0.483	0.873	0.596	0.912	0.388	0.590	0.461	0.660	0.307	0.323	0.354	0.371
45	0.587	0.959	0.660	0.964	0.502	0.913	0.568	0.927	0.397	0.586	0.448	0.638	0.306	0.317	0.350	0.363
50	0.562	0.923	0.691	0.952	0.486	0.881	0.586	0.914	0.388	0.589	0.456	0.654	0.301	0.314	0.352	0.366

TABLEAU 5. I. Efficiency of the MOT, MOT+LUBC, MOTe, MOTe+LUBC approaches in terms of the proportion of ordering of pairs of elements solved, on sets of uniformly distributed random sets of m permutations, $m = 3$ and $m = 5x$, $1 \leq x \leq 10$, statistics generated over 80000 sets for $n = 15$, 50000 sets for $n = 20$, 10000 sets for $n = 30$ and 1000 sets for $n = 45$.

5.6. An exact approach

Our third approach is an exact branch-and-bound solver for the problem that combines the result of the simulated annealing method of Section 5.4 with the constraints obtained in Section 5.5.4 to avoid exploring not promising search sub-trees.

5.6.1. Branch-and-Bound

Our branch-and-bound algorithm simply constructs possible medians of $\mathcal{A} \subseteq \mathbb{S}_n$, *i.e.* permutations of $[n]$ by putting a new element to the right of the already known ones till no more element are available. Thus we explore a tree having the empty permutation at its root, permutations of any k elements of $[n]$ as nodes of level k and where the leaves are permutations of $\mathbb{S}_n$. Each node N will have a corresponding lower bound $Lb(N)$ representing the fact that for all permutations π derived from N, $d_{KT}(\pi,\mathcal{A}) \geq Lb(N)$.

If this lower bound $Lb(N)$ is higher than the current upper bound of the problem, then all the permutations derived from it will have a higher distance than one already found and node N with all its descendants can be omitted in the exploration. As the number of nodes/leaves is finite and the bounding method only cuts branches that do not represent possible medians, the BnB will always converge to the optimal solution.

More specifically, given our set $\mathcal{A} \subseteq \mathbb{S}_n$, we run the SA heuristic of Section 5.4 to have a first approximative median of $\mathcal{A}$, π_{approx}, and a first upper bound $Ub_{\mathcal{A}}$, which will always represents the score associated with the currently best solution found. The approximative

solution will serve as guidance so that the first leaf that will be visited by the BnB will be π_{approx}. This guarantees us an efficient cutting of non-promising branches.

A node at level k is represented by a vector of size k, $x = [x_1, ..., x_k]$ which corresponds to a permutation in construction. Let S be the set of elements still to be placed *i.e* $S = [n] - \{x_1, x_2, ..., x_{k-1}, x_k\}$ and L a list which orders S. At the beginning of our BnB, $S = [n]$ and $L = \pi_{approx}$. The BnB will branch by choosing the next element from the list : $\ell_i \in L$ that will be placed at the immediate right of x_k. We are going to apply the bound and cuts $Bound_1$, Cut_1, Cut_2 and Cut_3 described in Section 5.6.2 below for each choice of ℓ_i. If it succeeds passing all the bounds test, $x_{k+1} \coloneqq \ell_i$, and we go down to the new node $x' = [x_1, x_2, ..., x_{k-1}, x_k, x_{k+1}]$. If not, we try ℓ_{i+1} as a possible x_{k+1}. If we go down to a leaf, *i.e.* if $k + 1 = n$, its corresponding permutation π_{leaf} is compared to the best solution. If $d_{KT}(\pi_{leaf}, \mathcal{A}) = Ub_{\mathcal{A}}$, we add π_{leaf} to the current set of medians. If $d_{KT}(\pi_{leaf}, \mathcal{A}) < Ub_{\mathcal{A}}$, then it is our new upper bound and we change our set of medians so that it contain only this new optimal permutation π_{leaf}. The BnB backtracks after all possibilities of $\ell_i \in L$ for x_{k+1} had been explored or after investigating a leaf node.

5.6.2. Bounds and Cuts

Now, let us set everything that is needed to described our different bound and cuts. First, given a set of permutations $\mathcal{A} \subseteq \mathbb{S}_n$, we deduce a valid set of constraints $C(\mathcal{A})$ using the method MOTe+LUBC described in Section 5.5.4. We also pre-calculated, for each triplet of elements x,y and $z \in [n]$ the best ways to put them all together, one after the other, in a median of $\mathcal{A}$. All the other ways to order them consecutively in a permutation will be called a forbidden triplet, since a permutation containing this ordering cannot be a median of $\mathcal{A}$.

For each node $x = [x_1, ..., x_k]$, we can compute a distance $d(x, \mathcal{A})$ in the following way :

$$d(x,\mathcal{A}) = \underbrace{\sum_{i=1}^{|x|} \sum_{j=i+1}^{|x|} R_{x_i x_j}(\mathcal{A})}_{\substack{\text{contribution to the Kendall-}\tau\text{ distance for} \\ \text{the elements already placed}}} + \underbrace{\sum_{i=1}^{|x|} \sum_{j=1}^{|L|} R_{x_i l_j}(\mathcal{A}).}_{\substack{\text{contribution obtained by the fact that all elements} \\ \text{of } x \text{ are to the left of elements in } L \text{ (yet to be placed)}}}$$

Finally, for each node x, let $b(x)$ be the boolean vector of length n representing which elements of $[n]$ are already placed in this node (*i.e* we have $b_i(x) = 1$, $1 \leq i \leq n$, if and only if $i = x_j$, $1 \leq j \leq k$). So if a node x' contained the same k elements of node x but in a different order, $b(x) = b(x')$. Our BnB will construct and update a set $TopScores$ of pairs $< b, v >$, where b is any boolean vector of length n and $v = \min d(x, \mathcal{A})$, for x already

explored such that $b(x) = b$.

Now, if our current best solution is π_{best}, our current upper bound is $Ub_{\mathcal{A}} = d_{KT}(\pi_{best},\mathcal{A})$, our current node is $x = [x_1, ..., x_k]$, L is an ordered list of the elements still to be placed, and we are studying $\ell_i \in L$ as a possible x_{k+1}, we have that :

- **Cut$_1$** : If (x_{k-1},x_k,ℓ_i) is a forbidden triplet then x_{k+1} cannot be l_i and so it is rejected. (*i.e* we do not explore the subtree having node $x = [x_1, ..., x_k, \ell_i]$ as its root.)
- **Cut$_2$** : If there exist $\ell_j \in L$, $\ell_j \neq \ell_i$ such that $C(\mathcal{A})(\ell_j,\ell_i) = 1$ then we know that ℓ_i has to be to the right of ℓ_j in a median permutation of $\mathcal{A}$ and so x_{k+1} cannot be ℓ_i and is rejected.
- **Cut$_3$** : Let $x' = [x_1, \ldots, x_k, \ell_i]$. If $< b(x'), v > \in TopScores$ and $d(x',\mathcal{A}) > v$ then x_{k+1} cannot be ℓ_i and so it is rejected.
- **Bound$_1$** : Let a lower bound of a list L be $lb(L) = \sum_{i=1}^{|L|} \sum_{j=i+1}^{|L|} (min\{R_{l_i l_j}, R_{l_i l_j}\}) + tri(L)$, where $tri(L)$ is a simpler implementation of the lower bound using only 3-cycles described in Section 4.2. In this implementation, the 3-cycles are pre-calculated at the beginning, and the contribution of a cycle is added if and only if all three of its elements are in L. Let $x' = [x_1, \ldots, x_k, \ell_i]$ and let L' be the list L without ℓ_i. Let $Lb(x') = d(x',\mathcal{A}) + lb(L')$. If $Lb(x') > Ub_{\mathcal{A}}$ then x_{k+1} cannot be ℓ_i and so it is rejected.

Algorithm 4, in Appendix C, gives the pseudo-code of our BnB method. As a final note, our BnB can solve in reasonable time (a few seconds in average) any problem with $n \leq 38, m \leq 50$, since we kept all calculation in linear time at the node level for efficiency purpose. The case where $m = 4$ is the hardest case (and the most variable in execution time) for the BnB, opposite to SA, as the average number of medians of $\mathcal{A} \subseteq \mathbb{S}_n$ have been observed to be the biggest for all m.

In [6], Ali and Meilă made a thorough comparison of many solvers and heuristics, solving uniformly generated problems of size up to $n = 50, m = 100$. We did some quick testing of our BnB on similar problems (same n, m and uniformly generated sets) and we claim that it solves them in a comparable time, thus competing with the best solvers (BnB and Integer Linear Programming). More intensive testing will be done in the near futur.

5.7. Conclusion

In this article, we studied the problem of finding the median of a set of m permutations $\mathcal{A} \subseteq \mathbb{S}_n$ under the Kendall-τ distance. This problem is known to be NP-hard for $m \geq 4$, m even. This work presents three different solving techniques for this problem ; a well parameterized simulated annealing heuristic, a space reduction technique and an promising exact BnB solver.

Ideas for future works includes an extensive comparison with other exact solvers and heuristics, as well as testing on various synthetic and real life data sets. It would also be interesting to take into account the fact that the rankings considered are not always on the entire sets of elements involved. Furthermore, some ranking schemes often rank several elements in the same position, so rank ties are to be considered.

5.8. Acknowledgements

We would like to thanks our anonymous reviewers for their careful and inspiring comments. Be sure that the suggestions that were not included here, due to time and space constraints, will be integrate in the journal version of this article.

Exploring the Median of Permutations Problem

Résumé

Cet article [99] est la version allongée de l'article présenté à la conférence IWOCA2017. Il a été accepté au Journal of Discrete Algorithms en octobre 2018.

Outre le contenu de l'article de conférence [97] (heuristique du recuit simulé, réduction d'espace et algorithme branch-and-bound), les ajouts suivants ont été effectués :

— de nombreux résultats de tests statistiques, supportant les choix de paramètres de l'heuristique du recuit simulé, ont été ajoutés,
— une combinaisons de notre réduction d'espace de recherche avec celle de [43] a été étudiée,
— les détails de l'algorithme branch-and-bound et de ses coupes ont été élaborés davantage à l'aide de pseudo-codes,
— l'implémentation de la programmation en nombre entiers (ILP) avec CPLEX pour résoudre le problème est discutée,
— des tests sont présentés pour montrer l'avantage d'utiliser l'heuristique du recuit simulé et la reduction d'espace couplé avec le modèle ILP,
— des tests sont effectués pour les quatre approches sur des données réelles,
— des comparaisons sont effectuées pour les quatre approches avec des méthodes récentes : il en sort que toutes nos approches sont compétitives.

À la fin nous suggérons que cette combinaison de méthode pourrait être la façon optimale de résoudre le problème de la médiane de permutations.

Partage du travail

Dans les contributions de cet article, Robin Milosz a implémenté, testé et paramétré l'heuristique du recuit simulé. Robin Milosz et Sylvie Hamel ont développé et prouvé la réduction d'espace de la seconde approche. Robin Milosz a généré les statistiques sur l'efficacité de cette méthode. Robin Milosz a développé et implémenté l'algorithme exact du branch-and-bound. Robin Milosz a implémenté le modèle de programmation en nombre entiers avec les combinaisons de méthodes. Les implémentations sont dans le langage Java. Tous les auteurs ont rédigé l'article. L'article se trouve à l'annexe B de cette thèse.

Chapitre 6

MÉDIANE DE 3 PERMUTATIONS

Résumé

L'article "Median of 3 Permutations, 3-Cycles and 3-Hitting Set Problem" a été présenté à la conférence IWOCA2018 et a été le fruit du travail réalisé dans le cadre du stage Mitacs au Laboratoire de Recherche en Informatique de l'Université Paris-Sud (Orsay).

Dans ce travail [**100**], le cas particulier du problème de la médiane de trois permutations étudié en profondeur. Il en ressort le théorème des 3-cycles qui permet d'ordonner toutes les paires d'éléments qui n'appartiennent pas à un 3-cycle dans le graphe de majorité relié aux trois permutations. Il est prouvé ensuite que cette méthode généralise 4 autres méthodes antérieures dans le cas de trois permutations. Des tests statistiques soutiennent l'utilité de cette approche. Une conjecture est émise sur la généralisation de ce théorème au cas de la médiane de m permutations, m impair.

Dans un deuxième temps, une importante deuxième conjecture est présentée. Elle relie le problème de la médiane de trois permutations (M3P) avec le problème du 3-Hitting Set (3HS). Plus précisément, on conjecture que résoudre le problème M3P équivaut à résoudre le problème 3HS, qui malgré le fait qu'il est NP-Difficile, est relativement plus facile à résoudre pour une taille d'entrée similaire. De plus, on montre que le problème 3HS donne une borne inférieure très stricte au problème M3P.

Finalement, des simulations massives ont été réalisées pour tenter de trouver un contre-exemple pour toutes les conjectures présentées, sans succès.

Partage du travail

Dans les contributions de cet article, Robin Milosz, Adeline Pierrot et Sylvie Hamel ont développé et prouvé le théorème des 3-cycles. Robin Milosz a émis les deux conjectures. Robin Milosz a testé l'efficacité de ces approches et généré les statistiques. Tous les auteurs ont rédigé l'article. Robin Milosz et Sylvie Hamel ont fait les corrections et soumis la version finale.

Median of 3 Permutations, 3-Cycles and 3-Hitting Set Problem

Robin Milosz[1,2] and Adeline Pierrot[2] and Sylvie Hamel[1]

[1] *Département d'Informatique et de Recherche Opérationnelle - Université de Montréal*
[2] *Laboratoire de Recherche Informatique - Université Paris-Sud*

6.1. Abstract

The median of permutations problem consists in finding a consensus permutation of a given set of m permutations of size n. This consensus represent the "closest" permutation to the given set under the Kendall-tau distance. Since the complexity of this problem is still unknown for sets of 3 permutations, in the following work, we investigate this specific case and show an interesting link with the 3-Hitting Set problem.

6.2. Introduction

The problem of aggregating multiple rankings into one consensus ranking was already looked at two centuries ago [43] but was mostly studied in the last twenty years under different names and in different research areas : *rank aggregation problem* [2, 56, 60], *Kemeny rank aggregation* [6, 18, 44], *Kemeny-Young method* [140], *median ranking problem* [40] and *preference aggregation* [51]. Applications include determining the winner in a sport competition, deriving voting preferences for an election or aggregating answers returned by several Web engines.

From a theoretical point-of-view, if the rankings are on strictly ordered elements, rankings are *permutations* and the problem becomes that of finding the *median* of a set of permutations under a given distance. For the Kendall-tau distance [78], the problem of finding medians of a set of m permutations has been widely studied either by deriving some exact solvers [6, 44], some approximation algorithms [117], some fixed-parameters algorithms [20, 75, 104] or working on space reduction techniques [18, 26, 96].

The problem has been proved to be NP-complete for sets of $m \geq 4$ permutations, m even [56] (some corrections of the proof was done in [23]). It was recently found [11] that it is NP-Hard for sets of $m \geq 7$ permutations, m odd. The theoretical complexity of the cases $m = 3$ and $m = 5$ remains open, thus making it relevant to investigate those two cases.

Here, we focus on the case where $m = 3$, i.e. we are interested to find medians of sets of three permutations. We demonstrate in this work, that in order to solve the median of three permutations problem, one can only consider 3-cycles present in the majority graph associated with the three permutations. We further make the link with the 3-Hitting Set problem (one of Karp's 21 NP-complete problems [74]) which consist in finding a minimal set of elements that cover a collection of subsets, in our case covering the collection of all 3-cycles present in the majority graph.

6.3. Basic Definitions

A *permutation* π is a bijection of $[n] = \{1,2\ldots,n\}$ onto itself. The set of all permutations of $[n]$ is denoted $\mathcal{S}_n$. As usual we denote a permutation π of $[n]$ as $\pi = \pi_1\pi_2\ldots\pi_n$. The *position* of an element i in a permutation π is π_i^{-1}, where π^{-1} is the usual inverse of π under composition. Let $\mathcal{R} \subset \mathcal{S}_n$ be a set of permutations of $[n]$, we will denote its *cardinality* by $\#\mathcal{R}$. We define the *order* between two elements i and j, $1 \leq i,j \leq n$, in a permutation π of $[n]$ as i *to the left of* j (resp. *to the right of*), denoted $i \prec_\pi j$ (resp. $j \prec_\pi i$), if $\pi_i^{-1} < \pi_j^{-1}$ (resp. $\pi_i^{-1} > \pi_j^{-1}$). For $\mathcal{R} \subset \mathcal{S}_n$, the *majority order* for elements i and j, $1 \leq i,j \leq n$ is $i \prec_\mathcal{R} j$ (resp. $j \prec_\mathcal{R} i$) if $\#\{\pi \in \mathcal{R} \mid i \prec_\pi j\} \underset{(resp.<)}{>} \#\{\pi \in \mathcal{R} \mid j \prec_\pi i\}$. We say that $j \prec_\mathcal{R} i$ is the *minority order* if $i \prec_\mathcal{R} j$ is the majority order. Note that when $\#\mathcal{R}$ is even, there can exist elements i and j, $1 \leq i,j \leq n$, for which neither a majority or a minority order is defined. The *Kendall-tau distance*, denoted Kt, counts the number of order disagreements between pairs of elements of two permutations $\pi, \sigma \in \mathcal{S}_n$ and is defined as $Kt(\pi,\sigma) = \#\{(i,j) | i < j \text{ and } [(i \prec_\pi j \text{ and } j \prec_\sigma i) \text{ or } (j \prec_\pi i \text{ and } i \prec_\sigma j)]\}$. Given any set of permutations $\mathcal{R} \subseteq \mathcal{S}_n$ and a permutation $\pi \in \mathcal{S}_n$, the *Kemeny score* is defined as $K(\pi,\mathcal{R}) = \sum_{\sigma\in\mathcal{R}} Kt(\pi,\sigma)$.

The *median of permutations* problem is stated as follows : Given $\mathcal{R} \subseteq \mathcal{S}_n$, we want to find a permutation $\pi^* \in \mathcal{S}_n$ such that $K(\pi^*,\mathcal{R}) \leq K(\pi,\mathcal{R})$, $\forall\ \pi \in \mathcal{S}_n$. Such a permutation π^* is called a *median permutation*. Note that a set $\mathcal{R}$ can have more than one median.

Finally, we define the *left matrice* $L(\mathcal{R})$ such that for any two elements $1 \leq i,j \leq n$, $L_{ij}(\mathcal{R}) = \#\{\pi \in \mathcal{R} \mid i \prec_\pi j\}$. It represents the number of time $i \prec j$ in permutations of $\mathcal{R}$. Note that $L_{xy}(\mathcal{R}) = \#\mathcal{R} - L_{yx}(\mathcal{R})$ and that the Kemeny score can be computed using the left matrice as $K(\pi,\mathcal{R}) = \sum_{1\leq i,j\leq n | i\prec_\pi j} L_{ji}(\mathcal{R})$. Note that for ease of notation, we will write L_{ij} instead of $L_{ij}(\mathcal{R})$, when the context is clear.

6.4. 3-cycle Theorem

In this section, we put everything together to state and prove our main theorem on the ordering of pairs of elements in medians of sets of three permutations.

6.4.1. Definitions and properties

First, let us set basic definitions and state some properties that will help understand and prove this main theorem called *3-cycle Theorem*. To do that, we need to build the following graph.

Definition 8. *Let $\mathcal{R} \subset \mathcal{S}_n$. The* **majority graph** *$G_{\mathcal{R}} = (V,E)$ of $\mathcal{R}$ is the weighted directed graph, with the set of vertices $V = \{i \,|\, 1 \leq i \leq n\}$ and the set of directed edges $E = \{(i,j) \,|\, i \prec_{\mathcal{R}} j$ is the majority order of $i \neq j \in V\}$. The* **weight** *of an edge (i,j) is $w_{(i,j)} = |L_{ij} - L_{ji}|$. Note that the majority graph is a weighted tournament graph when $\#\mathcal{R}$ is odd.*

Observation 1. *Let $\mathcal{R} \subset \mathcal{S}_n$ be a set of 3 permutations, then for every pair of elements i, j, $1 \leq i \neq j \leq n$, exactly one of the following holds :*

- *(i,j) is an edge of $G_{\mathcal{R}}$ of weight 3 (case $L_{ij} = 3$),*
- *(i,j) is an edge of $G_{\mathcal{R}}$ of weight 1 (case $L_{ij} = 2$),*
- *(j,i) is an edge of $G_{\mathcal{R}}$ of weight 1 (case $L_{ij} = 1$),*
- *(j,i) is an edge of $G_{\mathcal{R}}$ of weight 3 (case $L_{ij} = 0$).*

Definition 9. *We call* **LowerBound$_1$** *the trivial lower bound on the Kemeny Score of a median permutation of $\mathcal{R} \subset \mathcal{S}_n$, discussed in details in* [51]*, and computed as follow :*

$$LowerBound_1(K(\pi,\mathcal{R})) = \sum_{\substack{i<j \\ i\neq j\in\{1,\dots,n\}}} min\{L_{ij}(\mathcal{R}), L_{ji}(\mathcal{R})\}.$$

Definition 10. *A* **3-cycle** *(i,j,k) in a majority graph $G = (V,E)$ is a directed cycle of length three containing the edges (i,j),(j,k) and (k,i). The* **set of involved edges** *of G, denoted I_E, is the set of all edges in E that are contained in at least one 3-cycle.*

Example 4. *Figure 6.1 shows the majority graph of the set $\mathcal{R} = \{[4,5,1,2,3], [1,5,3,4,2], [5,2,3,4,1]\}$. In this instance, there are two 3-cycles : $(2,3,4)$ and $(3,4,1)$ and $I_E = \{(1,3),(2,3),(3,4),(4,1),(4,2)\}$.*

Proposition 6.4.1. *Let $\mathcal{R} \subset \mathcal{S}_n$ be a set of 3 permutations and let $G_{\mathcal{R}} = (V,E)$ be its majority graph. Then if $e \in E$ is an edge of a 3-cycle of $G_{\mathcal{R}}$, $w_e = 1$.*

Proof. By contradiction, suppose that there is an edge (i,j) of weight 3 in a 3-cycle (i,j,k) of $G_{\mathcal{R}}$. Then in every permutation of $\mathcal{R}$, i is before j. Moreover, by definition

of the majority graph, $k \prec_{\mathcal{R}} i$ and $j \prec_{\mathcal{R}} k$ have to be majority orders. Because $k \prec_{\mathcal{R}} i$ is a majority order, k has to be before i in at least 2 of the 3 permutations of $\mathcal{R}$. But i is always before j thus, by transitivity, k has to be before j in at least 2 of the 3 permutations. This is a contradiction because $j \prec_{\mathcal{R}} k$ is a majority order. Therefore, a 3-cycle can only have edges of weight 1. ■

Finding a median for a set $\mathcal{R} \subset \mathcal{S}_n$ of 3 permutations is equivalent to transforming the majority graph $G_{\mathcal{R}}$ into a direct acyclic graph (DAG) using the minimum number of edge inversions and taking into account the weights of those inverted edges [44, 51]. Indeed, a permutation can be represented as a DAG (the majority graph of the set containing only this permutation), and all edges that are reversed with respect to $G_{\mathcal{R}}$ correspond to pairs of elements that are in their minority order in that permutation. Note that if the number of reversed edges is minimal, the resulting topological ordering of the graph is a median permutation. From [44], we have that the Kemeny score of a permutations is linked with $LowerBound_1$ and the weight of reversed edges with respect to $G_{\mathcal{R}}$:

Proposition 6.4.2. [44] *Let $\mathcal{R} \subset \mathcal{S}_n$, let $G_{\mathcal{R}} = (V,E)$ be the majority graph of $\mathcal{R}$ and let $D(G_{\mathcal{R}})$ be a directed acyclic graph obtained by inverting edges in $G_{\mathcal{R}}$. Let π be the permutation obtained from the topological ordering of the vertices in $D(G_{\mathcal{R}})$, then :*

$$K(\pi,\mathcal{R}) = LowerBound_1 + \sum_{\substack{(i,j)\in E \\ s.t.(j,i)\in D(G_{\mathcal{R}})}} w_{(i,j)}.$$

Interestingly, we observed by simulations on sets of 3 permutations, that all edges of the majority graph that needed to be inverted to obtain a median permutation were contained in a 3-cycle of the original majority graph. This gives us the following theorem.

Théorème 6.4.1. (3-cycle Theorem) *Let $\mathcal{R} \subset \mathcal{S}_n$ be a set of 3 permutations. Let $G_{\mathcal{R}} = (V,E)$ be its majority graph. Let π^* be any median permutation of $\mathcal{R}$. If an edge (i,j) of $G_{\mathcal{R}}$ is not contained in any 3-cycle, then $i \prec_{\pi^*} j$.*

Proof (Part 1). First, note that by Proposition 6.4.1, all edges (i,j), with $w_{(i,j)} = 3$ are not contained in any 3-cycle of $G_{\mathcal{R}}$. For these edges, the following theorem from [26], gives us the result.

Théorème 6.4.2. (Always Theorem [26]**)** *Let $\mathcal{R}$ be a set of m permutations, m odd. Let π^* be any median permutation of $\mathcal{R}$. If $i \prec_{\pi} j$, $\forall \pi \in \mathcal{R}$, then $i \prec_{\pi^*} j$.*

For those edges (i,j) of $G_{\mathcal{R}}$, not contained in any 3-cycle, with $w_{(i,j)} = 1$, we have to work a bit more. The proof of this case is detailed in Section 6.4.2. ■

Theorem 6.4.1 can drastically reduce the search space for a median permutation, since it allows us to fix the order of all pairs of elements not contained in any 3-cycle of the majority graph. In Figure 6.1, all 3-cycles are represented with dashed edges.

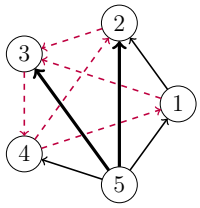


FIGURE 6.1. The majority graph of the set $\mathcal{R} = \{[4,5,1,2,3],[1,5,3,4,2],[5,2,3,4,1]\}$. Bold edges have weight 3 and thin edges have weight 1. The 3-cycles are shown with dashed edges and the set of all those dashed edges is the set of involved edges I_E. With the 3-cycle theorem, the edge $(1,2)$ is preserved in any median permutation and should not be reversed despite being part of the cycle $(1,2,3,4)$. In this case, reversing edge $(3,4)$ results in a DAG with corresponding median permutation $\pi^* = [5,4,1,2,3]$.

6.4.2. Proof of the 3-cycle Theorem

In the following section we prove Theorem 6.4.1 for the case where (i,j) of $G_{\mathcal{R}}$ is not contained in any 3-cycle, with $w_{(i,j)} = 1$, and where $\mathcal{R} \subset \mathcal{S}_n$ is a set of 3 permutations.

To help us do so, first note that we have the following partition.

Lemme 6.4.1. *Let $\mathcal{R} \subset \mathcal{S}_n$ be a set of 3 permutations and let $G_{\mathcal{R}} = (V,E)$ be its majority graph. Let (i,j) be an edge of $G_{\mathcal{R}}$ not contained in any 3-cycle. Then the elements of $\mathcal{N}_{(i,j)} = \{1,2,\ldots,n\} \setminus \{i,j\}$ can be partitioned into the 5 distincts (disjoint) following subsets A, B, C, D and E :*

$$\begin{aligned}
A &= \{x \in \mathcal{N}_{(i,j)} \,|\, L_{xi}(\mathcal{R}) = 3\},\\
B &= \{x \in \mathcal{N}_{(i,j)} \,|\, L_{xi}(\mathcal{R}) = 2 \text{ and } L_{xj}(\mathcal{R}) \geq 2\},\\
C &= \{x \in \mathcal{N}_{(i,j)} \,|\, L_{xi}(\mathcal{R}) \leq 1 \text{ and } L_{xj}(\mathcal{R}) \geq 2\},\\
D &= \{x \in \mathcal{N}_{(i,j)} \,|\, L_{xi}(\mathcal{R}) \leq 1 \text{ and } L_{xj}(\mathcal{R}) = 1\},\\
E &= \{x \in \mathcal{N}_{(i,j)} \,|\, L_{xj}(\mathcal{R}) = 0\}.
\end{aligned}$$

Proof. This comes from the fact that for $x \in \mathcal{N}_{(i,j)}$, we cannot have $L_{xi}(\mathcal{R}) \geq 2$ ($x \prec_{\mathcal{R}} i$ is a majority order) and $L_{xj}(\mathcal{R}) \leq 1$ ($j \prec_{\mathcal{R}} x$ is a majority order), since this case correspond to having the 3-cycle (x,i,j) and we choose an edge (i,j) of $G_{\mathcal{R}}$ not contained in any 3-cycle.

Since all of the other disjoint cases are represent by one of the subsets A, B, C, D and E, this conclude the proof. ■

Figure 6.2 gives two sets of three permutations, with elements in A, B, C, D and E.

$$\mathcal{R}_1 = \left\{ \begin{array}{cccccccccc} [& b & a & \mathbf{i} & & & c & & \mathbf{j} & e & d &] \\ & [& a & \mathbf{i} & d & & c & b & \mathbf{j} & e &] & \\ & & [& \mathbf{j} & e & d & c & b & a & \mathbf{i} &] & \end{array} \right\} \quad \mathcal{R}_2 = \left\{ \begin{array}{cccccccccc} [& a & b & \mathbf{i} & & & c & & \mathbf{j} & d & e &] \\ & [& a & \mathbf{i} & b & & c & d & \mathbf{j} & e &] & \\ & & [& \mathbf{j} & a & b & c & d & e & \mathbf{i} &] & \end{array} \right\}$$

FIGURE 6.2. Two examples of a set of three permutations having one element of each subset : $a \in A, b \in B, c \in C, d \in D$ and $e \in E$. The permutations are formatted to align elements i and j for better visualization.

To prove Theorem 6.4.1 by contradiction, we assume that there are two elements i and j such that (i,j) is an edge of the majority graph of $\mathcal{R}$ not contained in any 3-cycle, but there is a median permutation π^* in which $j \prec_{\pi^*} i$. We will concentrate on the set K of elements lying strictly between j and i in π^* (i.e. $\pi^* = ...jKi...$) and show that swapping j and i reduces the Kemeny score leading to a contradiction in our choice of median. Our proof is based on the following two lemmas, where Lemma 6.4.2 gives the possible values of $L_{xy}(\mathcal{R})$, for $x,y \in K \cup \{i,j\}$, and Lemma 6.4.3 shows that for $k \in K$, $k \notin A$ and $k \notin E$.

Lemme 6.4.2. *Let $\mathcal{R} \subset \mathcal{S}_n$ be a set of 3 permutations and let $G_{\mathcal{R}} = (V,E)$ be its majority graph. Let (i,j) be an edge of weight $w_{(i,j)} = 1$ in $G_{\mathcal{R}}$ not contained in any 3-cycle. Let $j \prec_{\pi^*} i$, where $\pi^* \in \mathcal{S}_n$ is an assumed median of $\mathcal{R}$. Let $K \subset \{1,2,\dots n\}$ be the set of elements lying strictly between j and i in π^*. Then, the values $L_{xy}(\mathcal{R})$ for $x,y \in K \cup \{i,j\}$ are the ones given in Table 6. I :*

L_{xy}	$y \in A$	$y \in B$	$y \in C$	$y \in D$	$y \in E$	$y = i$	$y = j$
$x \in A$	***0***,*1,2,3*	**1**,*2,3*	**2**,*3*	**2**,*3*	**2**,*3*	***3***	***2***
$x \in B$	*0,1*,**2**	***0***,*1,2,3*	**1**,*2,3*	**1**,*2,3*	**2**,*3*	***2***	***2***
$x \in C$	*0*,**1**	*0,1*,**2**	***0***,*1,2,3*	**1**,*2,3*	**2**,*3*	***1***	***2***
$x \in D$	*0*,**1**	*0,1*,**2**	*0,1*,**2**	***0***,*1,2,3*	**1**,*2,3*	***1***	***1***
$x \in E$	*0*,**1**	*0*,**1**	*0*,**1**	*0,1*,**2**	***0***,*1,2,3*	***1***	***0***
$x = i$	***0***	***1***	***2***	***2***	***2***	-	***2***
$x = j$	***1***	***1***	***1***	***2***	***3***	***1***	-

TABLEAU 6. I. This table gives the possible values of $L_{xy}(\mathcal{R})$ for $x,y \in K \cup \{i,j\}$ (i.e. the number of permutations of $\mathcal{R}$ having x to the left y), where K is partitioned into the subsets A, B, C, D and E of Lemma 6.4.1. The bold (resp. italicized) numbers corresponds to values of $L_{xy}(\mathcal{R}_1)$ (resp. of $L_{xy}(\mathcal{R}_2)$) of the left example from Figure 6.2.

Proof. First, Theorem 6.4.2 and the fact that $\pi^* = \dots jKi \dots$ is a median of $\mathcal{R}$ imply that $L_{kj}(\mathcal{R}) \neq 3$ and $L_{ik}(\mathcal{R}) \neq 3, \forall k \in K$. Combining this observation and the definitions of subsets A, B, C, D, E, give us that $L_{ai} = 3$, $\forall a \in A$, $L_{bi} = L_{bj} = 2$, $\forall b \in B$, $L_{ic} = L_{cj} = 2$, $\forall c \in C$, $L_{id} = L_{jd} = 2$, $\forall d \in D$, and $L_{je} = 3$, $\forall e \in E$.

Second, since (i,j) is an edge of weight $w_{(i,j)} = 1$ in $G_{\mathcal{R}}$, we know that the majority order between i and j is $i \prec_{\mathcal{R}} j$ with $L_{ij}(\mathcal{R}) = 2$. Since $\forall a \in A$, a is always left of i, by transitivity, every element $a \in A$ has the majority order $a \prec_{\mathcal{R}} j$ and $L_{aj} = 2$, $\forall a \in A$. Transitivity also gives us $L_{ak} \geq 2$, $\forall a \in A$ and $\forall k \in C \cup D \cup E$. Symmetrically, $L_{ie} = 2$, $\forall e \in E$ and $L_{ke} \geq 2$, $\forall e \in E$ and $\forall k \in A \cup B \cup C$.

Third, since $\forall b \in B$, $L_{bi}(\mathcal{R}) = 2$, we know that in one of the three permutations of $\mathcal{R}$, b is to the right of i. Combining this with the fact that $\forall a \in A$, a is always to the left of i ($L_{ai} = 3$) gives us that $L_{ba} \neq 3$, $\forall a \in A, b \in B$. Similarly, we have $L_{cb} \neq 3$, $\forall b \in B, c \in C$, $L_{dc} \neq 3$, $\forall c \in C, d \in D$, $L_{ed} \neq 3$, $\forall d \in D, e \in E$ and $L_{db} \neq 3$, $\forall b \in B, d \in D$.

Finally, if we consider x and y to be elements of the same subset A,B,C,D or E, then we know nothing about their relative order in permutations of $\mathcal{R}$ leading, in that case to $L_{xy} \in \{0,1,2,3\}$. ∎

Lemme 6.4.3. *Let $\mathcal{R} \subset \mathcal{S}_n$ be a set of 3 permutations and let $G_{\mathcal{R}} = (V,E)$ be its majority graph. Let (i,j) be an edge of weight $w_{(i,j)} = 1$ in $G_{\mathcal{R}}$ not contained in any 3-cycle. Let $\pi^* \in \mathcal{S}_n$ be an assumed median of $\mathcal{R}$ in which $j \prec_{\pi^*} i$. Let $K \subset \{1,2,\dots n\}$ be the set of elements lying strictly between j and i in π^*. Then, $\forall k \in K$, $L_{ki}(\mathcal{R}) \neq 3$ and $L_{kj}(\mathcal{R}) \neq 0$.*

Proof. First, note that if we partition K into the subsets A, B, C, D and E of Lemma 6.4.1, then for $k \in K$, showing that $L_{ki}(\mathcal{R}) \neq 3$ and $L_{kj}(\mathcal{R}) \neq 0$, is the same as showing that $k \notin A$ and $k \notin E$ i.e. subsets A and E are empty. To prove that A and E are empty, we will proceed in two steps. First, we will assume that they are not empty and show that we cannot have an element of E to the left of an element of A in our assumed median. Second, we will use this fact to conclude that A and E are empty.

Step 1. We want to show that there is no element of E to the left of an element of A in π^*. By contradiction, assume π^* has at least one element of E at the left of one element of A. Assume that $e \in E$ (resp. $a \in A$) is the rightmost (resp. leftmost) such element, so that there is neither elements of E nor elements of A lying between e and a in π^*. Let S be the sequence of elements lying between e and a in π^* (i.e. $\pi^* = ...j...eSa...i...$). Then S contains only elements of B, C and D. Let $B_s = B \cap S$ and $n_B = \#B_s$, and similarly for C and D. We show that we can build from π^* a permutation π' that reduces the Kemeny score.

If $n_D \geq n_B$ we take π' such that the position of every element in π' is the same as in π^*, except that a is moved to the left of S and $e : \pi' = ...j...aeS...i....$

Let Δ be the difference of Kemeny score between π' and $\pi^* : \Delta = K(\pi',\mathcal{R}) - K(\pi^*,\mathcal{R})$. Recall that for any permutation π, the Kemeny score $K(\pi,\mathcal{R}) = \sum_{1\leq i,j\leq n|\pi_i^{-1}<\pi_j^{-1}} L_{ji}(\mathcal{R})$. Then

$$\begin{aligned}
\Delta &= -L_{ae} + L_{ea} + \sum_{b\in B_s}(-L_{ab} + L_{ba}) + \sum_{c\in C_s}(-L_{ac} + L_{ca}) + \sum_{d\in D_s}(-L_{ad} + L_{da}) \\
&\overset{(1)}{<} -2 + 1 + \sum_{b\in B_s}(-1 + 2) + \sum_{c\in C_s}(-2 + 1) + \sum_{d\in D_s}(-2 + 1) \\
&= -1 + \sum_{b\in B_s}(1) + \sum_{c\in C_s}(-1) + \sum_{d\in D_s}(-1) \\
&= -1 + n_B - n_C - n_D \\
&\overset{(2)}{<} 0
\end{aligned}$$

Inequality (1) comes from taking the bounding values of L in Lemma 6.4.2. Inequality (2) comes from the fact that $n_D \geq n_B$.

For $n_B > n_D$, if we take π' such that the position of every element in π' is the same as in π^*, except that e is moved to the right of S and $a : \pi' = ...j...Sae...i...$, we can show again, using the same kind of computation as before, that $\Delta = K(\pi',\mathcal{R}) - K(\pi^*,\mathcal{R}) < 0$. In both cases, the Kemeny score is reduced, contradicting that π^* is a median permutation. Therefore, no element of E can precede an element of A in π^*.

Step 2. Now, we show that A and E are empty. Assume by contradiction that A is non empty. Let a be the leftmost element of A in π^*. Let S be the sequence of elements lying between j and a in π^* (i.e. $\pi^* = ...jSa...i...$). From Step 1, S contains only elements of B, C and D. Let $B_s = B \cap S$ and $n_B = \#B_s$, and similarly for C and D. Again we show that we can build from π^* a permutation π' that reduces the Kemeny score : If $n_B > n_D$ we take π' such that the position of every element in π' is the same as in π^*, except that j is moved to the right of S and $a : \pi' = ...Saj...i....$ Let Δ be the difference of Kemeny score between π' and $\pi^* : \Delta = K(\pi',\mathcal{R}) - K(\pi^*,\mathcal{R})$. Then

$$\begin{aligned}
\Delta &= -L_{aj} + L_{ja} + \sum_{b\in B_s}(-L_{bj} + L_{jb}) + \sum_{c\in C_s}(-L_{cj} + L_{jc}) + \sum_{d\in D_s}(-L_{dj} + L_{jd}) \\
&\overset{(1)}{=} -2 + 1 + \sum_{b\in B_s}(-2 + 1) + \sum_{c\in C_s}(-2 + 1) + \sum_{d\in D_s}(-1 + 2) \\
&= -1 + \sum_{b\in B_s}(-1) + \sum_{c\in C_s}(-1) + \sum_{d\in D_s}(1) \\
&= -1 - n_B - n_C + n_D \\
&\overset{(2)}{<} 0
\end{aligned}$$

Equality (1) comes from taking the values of L, given in Table 6. I. Inequality (2) comes from the fact that $n_B > n_D$.

For the case $n_D \geq n_B$, we move a at the immediate left of j (i.e. $\pi' = ...ajS...i...$) and we can show again, using the same kind of computation as before, that $\Delta = K(\pi',\mathcal{R}) - K(\pi^*,\mathcal{R}) < 0$. In both cases, the Kemeny score is reduced, contradicting that π^* is a median permutation. Therefore, A is empty. The proof that E is empty is symmetrical. ■

We now have everything we need to finish the proof of our Theorem 6.4.1 and conclude this section.

Proof of Theorem 6.4.1 (Part 2). Let $\mathcal{R} \subset \mathcal{S}_n$ be a set of 3 permutations and let $G_\mathcal{R} = (V,E)$ be its majority graph. Let (i,j) be an edge of weight $w_{(i,j)} = 1$ in $G_\mathcal{R}$ not contained in any 3-cycle. Let $\pi^* \in \mathcal{S}_n$ be an assumed median of $\mathcal{R}$ with $j \prec_{\pi^*} i$. Let $K \subset \{1,2,\dots n\}$ be the set of elements lying strictly between j and i in π^* i.e. we have $\pi^* = ...jKi....$ From Lemma 6.4.3, K contains only elements of B, C and D. Let $\pi' = ...iKj...$ obtained from π^* by exchanging the positions of element i and element j. Let Δ be the difference of Kemeny score between π' and π^*, i.e. $\Delta = K(\pi',\mathcal{R}) - K(\pi^*,\mathcal{R})$. Then

$$\begin{aligned}
\Delta &= -L_{ij} + L_{ji} + \sum_{b\in B}(-L_{ib} + L_{bi} - L_{bj} + L_{jb}) \\
&\quad + \sum_{c\in C}(-L_{ic} + L_{ci} - L_{cj} + L_{jc}) + \sum_{d\in D}(-L_{id} + L_{di} - L_{dj} + L_{jd}) \\
&\overset{(1)}{=} -2 + 1 + \sum_{b\in B}(-1 + 2 - 2 + 1) \\
&\quad + \sum_{c\in C}(-2 + 1 - 2 + 1) + \sum_{d\in D}(-2 + 1 - 1 + 2) \\
&= -1 + \sum_{b\in B}(0) + \sum_{c\in C}(-2) + \sum_{d\in D}(0) \\
&= -1 + -2 \times \#C \\
&< 0
\end{aligned}$$

Equality (1) comes from taking the values of L given in Lemma 6.4.2. The permutation π' has a lower Kemeny score than the assumed median permutation π^*, leading to a contradiction. ■

From preliminary tests we observed that Theorem 6.4.1 seems to be also valid on sets of m permutations, m odd. Therefore we state the following conjecture :

Conjecture 1. *Let $\mathcal{R} \subset \mathcal{S}_n$ be a set of m permutations, m odd. Let $G_\mathcal{R} = (V,E)$ be its majority graph. Let π^* be any median permutation of $\mathcal{R}$. If an edge (i,j) of $G_\mathcal{R}$ is not contained in any 3-cycle, then $i \prec_{\pi^*} j$.*

The 3-Cycle Theorem has an important significance since we can prove that pairs of elements that are ordered by previous methods such as an extension of the Condorcet

Criterion [43], the Major Order Theorem [96], the Always Theorem [26] and the 3/4 - Majority Rule [18], the two last ones being equivalent in the case of three permutations, are also ordered by Theorem 6.4.1. When combined with an ILP solver this space reduction greatly improves the solving time for randomly generated data as well as for real data (see supplementary material [1]).

6.5. Link with the 3-Hitting Set problem

The Hitting Set problem (HSP) is defined as follows : Let $\mathcal{E}$ be our set of elements and T a set of subsets of $\mathcal{E}$. Find the minimum cardinality subset $S \subseteq \mathcal{E}$ such that every subset of T contains at least one element of S.

The **3-Hitting Set problem (3HS)** is the case where all subsets in T are of size 3. Both the Hitting Set problem and the 3-Hitting Set problem are NP-Hard [74]. The 3-Hitting Set problem can be formulated in Integer Linear Programming (ILP) as :

$$minimize : \sum_{e \in \mathcal{E}} x_e$$

Subject to :

$$x_{e_{t1}} + x_{e_{t2}} + x_{e_{t3}} \geq 1, \quad \forall \{t_1,t_2,t_3\} \in T,$$
$$x_e \in \{0,1\}, \quad \forall e \in \mathcal{E}.$$

Where x_e is the binary variable that indicates if element e is included in the solution subset $S \subseteq \mathcal{E}$. The first constraint forces the solution to cover every subset $\{t_1,t_2,t_3\}$ of T with a least one element. The last constraint make the variables x_e binary.

Let S be the solution (the set of selected elements) of the 3-Hitting Set problem and $\#S$ its "value" as the score of the objective function, i.e here its cardinality.

The link between the 3-Hitting Set problem and the median of a set $\mathcal{R} \subset \mathcal{S}_n$ of three permutations is made by observing that for every 3-cycle of $G_{\mathcal{R}} = (V,E)$, at least one edge has to be reversed to obtained a permutation. Therefore, the minimum number of edges covering all 3-cycles is a lower bound on the number of edges needed to be reverse to make $G_{\mathcal{R}}$ a DAG. In order to cover all 3-cycles using the 3HS, we choose $\mathcal{E} = I_E$, the set of involved edges of the majority graph $G_{\mathcal{R}} = (V,E)$ (see Definition 10), and T, the set of 3-cycles of $G_{\mathcal{R}}$ (see Figure 6.3).

Proposition 6.5.1. *Let $\mathcal{R} \subset \mathcal{S}_n$ be a set of 3 permutations, $LowerBound_1$ be the trivial lower bound of Definition 9 and $G_{\mathcal{R}} = (V,E)$ be the majority graph of $\mathcal{R}$. Let T be the set*

1. http://www.iro.umontreal.ca/~hamelsyl/M3P_3Cycles_3HS.html

of all 3-cycles of $G_{\mathcal{R}}$ and I_E, the set of its involved edges. If S is a solution of value $\#S$ of the 3HS problem applied on (I_E,T) then :

$$LowerBound_1 + \#S \leq K(\pi^*,\mathcal{R}) \leq K(\pi,\mathcal{R}) \ \forall \pi \in S_n,$$

where π^* is a median permutation of $\mathcal{R}$.

Proof. From Theorem 6.4.1, only edges contained in 3-cycles can be reversed to make $G_{\mathcal{R}}$ a DAG with a minimal number of reversed edges. Since $\#S$ is the minimal number of edges to select in order to cover all sets of T in the 3HS, we have that $G_{\mathcal{R}}$ cannot become a DAG with less than $\#S$ reversed edges.

From Observation 1 and Proposition 6.4.1, we have that the cost of reversing the edge (i,j) of a 3-cycle in $G_{\mathcal{R}}$ is 1.

From Proposition 6.4.2, we have that the Kemeny Score of a permutation is the sum of the $LowerBound_1$ and the number of reversed edges in $G_{\mathcal{R}}$ times their weight. In the case of a median of three permutations, the weight of edges in 3-cycles is always 1. Then, any reversed edge contributes 1 to the Kemeny Score. Since $\#S$ is a lower bound on the number of reversed edges, we have that $LowerBound_1 + \#S$ is a lower bound for the Kemeny score of a median permutation. ■

This result greatly improves, for this $\#\mathcal{R} = 3$ case, the two lower bounds based on linear programming and cycles given in [44], which had to consider all cycles present in the majority graph.

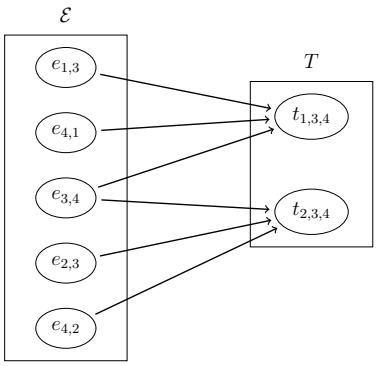


FIGURE 6.3. 3-Hitting Set problem with $\mathcal{E}$ as the set of involved edges I_E and T the set of 3-cycles from the example in Figure 6.1. In this case, selecting edge $e_{3,4}$ covers all 3-cycles thus $S = \{e_{3,4}\}$ and $\#S = 1$. Reversing this edge in the original majority graph makes the graph of Figure 6.1 acyclic.

Through preliminary empirical observations, we noted that the lower bound $LowerBound_1 + \#S$ seems to be a really tight lower bound : in all our tests we have the surprising observation of $LowerBound_1 + \#S = K(\pi^*,\mathcal{R})$, with π^* a median permutation of $\mathcal{R} \subset \mathcal{S}_n$, so we conjecture the following :

Conjecture 2. *Let $\mathcal{R} \subset \mathcal{S}_n$ be a set of 3 permutations, $LowerBound_1$ be the trivial lower bound of Definition 9 and $G_\mathcal{R} = (V,E)$ be the majority graph of $\mathcal{R}$. Let T be the set of all 3-cycles of $G_\mathcal{R}$ and I_E, the set of its involved edges. If S is a solution of value $\#S$ of the 3HS problem applied on (I_E,T) then :*

$$LowerBound_1 + \#S = K(\pi^*,\mathcal{R}) \leq K(\pi,\mathcal{R}) \ \forall \pi \in S_n,$$

where π^* is a median permutation of $\mathcal{R}$.

The conjecture was extensively tested on 10000 sets of 3 uniform random permutations of size $n \in \{10,20,30,40,50\}$ and 70 sets of permutations from PrefLib.org [91] (with $10 \leq n \leq 90$). No counter-example has been found. The conjecture can be extended for the cases where $\mathcal{R} \subset \mathcal{S}_n$ is a set of m permutations, m odd, with the weighted 3-Hitting Set problem.

6.6. Perspectives

The theoretical complexity of the median of permutations problem on sets of three permutations is still unknown making it an interesting problem to investigate. In this article, we presented our 3-Cycle Theorem, that allows us to drastically reduce the search space for a median of 3 permutations by fixing the order of all pairs of elements not contained in any 3-cycle of the majority graph of the given set of permutations. We then conjecture that our 3-Cycle Theorem is still true for sets of m permutations, m odd. It would be very interesting to be able to prove the stated conjecture since its reach englobes previous methods such as the Extended Condorcet Criterion [43], the Major Order Theorem [96], the Always Theorem [26] and the 3/4 - Majority Rule [18].

We also stated a new tight lower bound on the problem and a conjecture linking the 3HS problem with the median of three permutations problem. In future works, it would be interesting to prove this conjecture and to investigate the complexity of the median of 3 permutations problem using this link to the 3-Hitting Set problem which is known to be NP-Hard.

6.7. Acknowledgements

Thanks to Sarah Cohen-Boulakia, Alain Denise and Pierre Andrieu from the bioinformatic team of Laboratoire de Recherche Informatique of Université Paris-Sud for useful advices and thoughts. Thanks to Mitacs which made this collaboration possible through a Mitacs Globalink grant.

Median of 3 Permutations, 3-Cycles and 3-Hitting Set Problem

Robin Milosz[1,2] and Adeline Pierrot[2] and Sylvie Hamel[1]

[1] *Département d'Informatique et de Recherche Opérationnelle - Université de Montréal*

[2] *Laboratoire de Recherche Informatique - Université Paris-Sud*

6.8. Supplementary material

Supplementary material for Median of 3 Permutations, 3-Cycles and 3-Hitting Set Problem

Here we present results on real and randomly generated data as supplementary material for the "Median of 3 Permutations, 3-Cycles and 3 Hitting Set Problem" article. Tables 6. II and 6. III show results on real data from PrefLib.org [**91**]. Tables 6. V and 6. VI show results on randomly uniformly generated data. In all cases, Conjecture 2 is holding.

A clear distinction can be seen between those two sources of data : real life data permutations have much more similarity than the randomly uniformly generated ones. Therefore, our approach has a stronger impact on real life data. It is also easier to compute the median permutation for real life data.

For real life data from PrefLib (Tables 6. II and 6. III), we have that, in average, 87.7% of all the pairs of elements were ordered by the 3-cycle theorem. Also, in average, only 3.2% of the maximal number of 3-cycles were present pointing to the efficiency in term of time complexity of our new approach. Table 6. III shows effectively that using the 3-cycle theorem constraints with the Integer Linear Programming (ILP) model is, on average, 3.7 time faster then solving the original ILP model and solving the 3-Hitting Set ILP model is, on average, 187.1 time faster then solving the original ILP model.

For uniformly generated random sets (Tables 6. V and 6. VI), we have that, in average, 48.5% of all the pairs of elements were ordered by the 3-cycle theorem. Also, in average, 20.4% of the possible 3-cycles were present, which is more than 6 times the quantity we had for real data. This affect our gain in term of time complexity. Table 6. VI shows effectively that using the 3-cycle theorem constraints with the Integer Linear Programming (ILP) model on random data is, on average, only 1.6 time faster then solving the original ILP model and solving the 3-Hitting Set ILP model is, on average, 19.3 time faster then solving the original ILP model.

The sets of permutations used for to compute all the statistics of Table 6. II to Table 6. VI can be all found at http://www.iro.umontreal.ca/~hamelsyl/M3P_3Cycles_3HS.html.

		A	B		C	D				A	B		C	D	
ID	n	# 3-cycles	max # 3-cycles	A/B	nb constr.	max nb constr.	C/D	ID	n	# 3-cycles	max # 3-cycles	A/B	nb constr.	max nb constr.	C/D
ex1	29	78	1015	7.7%	302	406	74.4%	ex36	44	255	3542	7.2%	675	946	71.4%
ex2	20	2	330	0.6%	185	190	97.4%	ex37	18	0	240	0%	153	153	100%
ex3	44	0	3542	0%	946	946	100%	ex38	17	23	204	11.3%	98	136	72.1%
ex4	64	124	10912	1.1%	1848	2016	91.7%	ex39	40	0	2660	0%	780	780	100%
ex5	24	0	572	0%	276	276	100%	ex40	73	203	16206	1.3%	2378	2628	90.5%
ex6	67	1116	12529	8.9%	1457	2211	65.9%	ex41	67	1203	12529	9.6%	1403	2211	63.5%
ex7	23	1	506	0.2%	250	253	98.8%	ex42	12	2	70	2.9%	61	66	92.4%
ex8	42	173	3080	5.6%	662	861	76.9%	ex43	18	6	240	2.5%	141	153	92.2%
ex9	28	36	910	4%	326	378	86.2%	ex44	45	14	3795	0.4%	962	990	97.2%
ex10	11	9	55	16.4%	39	55	70.9%	ex45	82	1444	22960	6.3%	2282	3321	68.7%
ex11	70	92	14280	0.6%	2283	2415	94.5%	ex46	26	9	728	1.2%	306	325	94.2%
ex12	67	873	12529	6.7%	1524	2211	68.9%	ex47	36	157	1938	8.1%	449	630	71.3%
ex13	63	63	10416	0.6%	1856	1953	95%	ex48	70	23	14280	0.2%	2380	2415	98.6%
ex14	23	0	506	0%	253	253	100%	ex49	81	1317	22140	6%	2303	3240	71.1%
ex15	43	4	3311	0.1%	894	903	99%	ex50	52	0	5850	0%	1326	1326	100%
ex16	21	10	385	2.6%	187	210	89%	ex51	89	1555	29370	5.3%	2796	3916	71.4%
ex17	14	0	112	0%	91	91	100%	ex52	18	0	240	0%	153	153	100%
ex18	23	0	506	0%	253	253	100%	ex53	68	1293	13090	9.9%	1423	2278	62.5%
ex19	40	0	2660	0%	780	780	100%	ex54	37	3	2109	0.1%	659	666	98.9%
ex20	52	314	5850	5.4%	1046	1326	78.9%	ex55	23	8	506	1.6%	236	253	93.3%
ex21	17	7	204	3.4%	122	136	89.7%	ex56	87	2064	27434	7.5%	2340	3741	62.6%
ex22	23	1	506	0.2%	250	253	98.8%	ex57	32	35	1360	2.6%	433	496	87.3%
ex23	72	6	15540	0.04%	2544	2556	99.5%	ex58	60	669	8990	7.4%	1207	1770	68.2%
ex24	20	6	330	1.8%	176	190	92.6%	ex59	41	94	2870	3.3%	697	820	85%
ex25	14	0	112	0%	91	91	100%	ex60	23	0	506	0%	253	253	100%
ex26	55	600	6930	8.7%	960	1485	64.6%	ex61	38	0	2280	0%	703	703	100%
ex27	77	1319	19019	6.9%	1978	2926	67.6%	ex62	30	0	1120	0%	435	435	100%
ex28	56	542	7308	7.4%	1032	1540	67%	ex63	19	0	285	0%	171	171	100%
ex29	32	8	1360	0.6%	481	496	97%	ex64	19	2	285	0.7%	166	171	97.1%
ex30	71	554	14910	3.7%	1963	2485	79%	ex65	20	1	330	0.3%	187	190	98.4%
ex31	41	162	2870	5.6%	636	820	77.6%	ex66	18	3	240	1.3%	145	153	94.8%
ex32	24	0	572	0%	276	276	100%	ex67	55	415	6930	6%	1088	1485	73.3%
ex33	18	10	240	4.2%	133	153	86.9%	ex68	21	22	385	5.7%	175	210	83.3%
ex34	69	0	13685	0%	2346	2346	100%	ex69	20	0	330	0%	190	190	100%
ex35	24	2	572	0.4%	271	276	98.2%	ex70	30	96	1120	8.6%	319	435	73.3%

TABLEAU 6. II. Statistics on real data from PrefLib.org[**91**]. Table 6. IV gives the correspondance between each example ID and the original file name of this example on the site PrefLib.org, as well as the number of permutations in that example. For each of them, 3 permutations were randomly drawn to compute the statistics of this table. For each example, n represents the length of the permutations, column A gives the number of 3-cycles found in this example, column B gives the maximal number of 3-cycles that could be present in an example of size n (see [**16**] for the formulas for odd and even length cases), column A/B gives the proportion, in %, of 3-cycles found compared to the maximum number of 3-cycles, column C gives the number of constraints $i \prec j$, $1 \leq i \neq j \leq n$ found, column D gives the maximal number of constraints to be found for permutations of size n and finally column C/D gives the % of constraints found.

		A	B		C				A	B		C	
ID	n	time ILP normal	time ILP 3 cyc.thm.	acc. factor A/B	time 3HS	acc. factor A/C	ID	n	time ILP normal	time ILP 3 cyc.thm.	acc. factor A/B	time 3HS	acc. factor A/C
ex1	29	57.033	23.992	2.4	1.507	37.8	ex36	44	204.593	82.487	2.5	4.353	47.0
ex2	20	17.791	4.582	3.9	0.285	62.4	ex37	18	12.574	2.312	5.4	0.041	304.0
ex3	44	192.875	35.615	5.4	0.241	799.8	ex38	17	10.927	4.135	2.6	0.534	20.5
ex4	64	622.724	192.885	3.2	4.915	126.7	ex39	40	145.282	26.675	5.5	0.199	728.9
ex5	24	30.349	5.614	5.4	0.072	419.6	ex40	73	931.125	269.027	3.5	6.509	143.1
ex6	67	995.763	388.635	2.6	28.514	34.9	ex41	67	854.930	425.523	2.0	46.834	18.3
ex7	23	27.296	7.602	3.6	0.369	74.0	ex42	12	3.659	0.956	3.8	0.107	34.1
ex8	42	179.002	64.705	2.8	3.311	54.1	ex43	18	12.958	3.310	3.9	0.261	49.7
ex9	28	50.616	15.729	3.2	0.999	50.7	ex44	45	212.428	57.061	3.7	1.573	135.1
ex10	11	2.792	1.120	2.5	0.142	19.7	ex45	82	1662.284	868.206	1.9	30.926	53.8
ex11	70	816.276	217.777	3.8	6.523	125.1	ex46	26	39.927	10.785	3.7	0.517	77.3
ex12	67	2525.575	551.535	4.6	67.437	37.5	ex47	36	113.438	46.559	2.4	2.693	42.1
ex13	63	594.110	158.109	3.8	5.285	112.4	ex48	70	813.056	202.603	4.0	4.308	188.7
ex14	23	26.629	4.925	5.4	0.067	400.2	ex49	81	1536.425	619.233	2.5	31.926	48.1
ex15	43	185.888	45.939	4.1	1.331	139.7	ex50	52	319.479	59.188	5.4	0.337	948.2
ex16	21	20.769	7.250	2.9	0.355	58.5	ex51	89	1781.754	725.566	2.5	27.735	64.2
ex17	14	5.808	1.061	5.5	0.026	225.5	ex52	18	12.571	2.311	5.4	0.041	304.0
ex18	23	26.614	4.925	5.4	0.067	400.0	ex53	68	823.985	409.850	2.0	23.489	35.1
ex19	40	145.265	26.685	5.4	0.199	728.8	ex54	37	118.024	29.106	4.1	0.974	121.2
ex20	52	570.203	162.058	3.5	6.082	93.8	ex55	23	27.455	7.360	3.7	0.428	64.2
ex21	17	10.856	4.129	2.6	0.294	36.9	ex56	87	5203.084	873.588	6.0	41.691	124.8
ex22	23	27.306	6.977	3.9	0.369	74.1	ex57	32	76.113	23.250	3.3	1.162	65.5
ex23	72	882.952	218.870	4.0	3.876	227.8	ex58	60	539.151	237.575	2.3	10.432	51.7
ex24	20	17.831	5.519	3.2	0.308	58.0	ex59	41	164.004	51.911	3.2	2.498	65.7
ex25	14	5.825	1.063	5.5	0.026	226.2	ex60	23	26.613	4.916	5.4	0.067	400.0
ex26	55	410.391	185.020	2.2	9.666	42.5	ex61	38	124.144	22.813	5.4	0.180	690.0
ex27	77	1154.598	499.389	2.3	22.834	50.6	ex62	30	60.214	11.105	5.4	0.112	535.8
ex28	56	445.740	192.977	2.3	9.181	48.6	ex63	19	14.818	2.734	5.4	0.046	323.0
ex29	32	75.622	19.030	4.0	0.796	95.0	ex64	19	15.238	4.489	3.4	0.258	59.0
ex30	71	865.348	319.743	2.7	11.106	77.9	ex65	20	17.816	4.976	3.6	0.279	64.0
ex31	41	166.500	72.933	2.3	4.831	34.5	ex66	18	12.928	3.963	3.3	0.237	54.6
ex32	24	30.347	5.614	5.4	0.072	419.6	ex67	55	405.842	161.717	2.5	6.858	59.2
ex33	18	12.944	4.682	2.8	0.346	37.4	ex68	21	22.442	8.917	2.5	0.553	40.6
ex34	69	748.569	139.640	5.4	0.594	1260.0	ex69	20	17.346	3.205	5.4	0.051	342.3
ex35	24	31.121	7.994	3.9	0.409	76.1	ex70	30	97.063	36.056	2.7	3.651	26.6

TABLEAU 6. III. Deterministic (CPLEX's ticks) time to solve each example of Table 6. II. For each example, n represents the length of the permutations, column A gives the time required to solve the Integer Linear Programming (ILP model of this example, column B gives the time required to solve the ILP model with constraints added from the 3-cycle theorem, column B/A gives the acceleration factor of ILP + 3-cycles with respect to ILP, column C gives the time required to solve the ILP model of the 3-Hitting Set Problem related to this example and finally, column C/A gives the acceleration factor of the 3-Hitting Set Problem with respect to the ILP model.

ID	PrefLib.org file name	m	ID	PrefLib.org file name	m
ex1	ED-00015-00000063.soc	4	ex36	ED-00015-00000056.soc	4
ex2	ED-00006-00000044.soc	9	ex37	ED-00006-00000035.soc	9
ex3	ED-00015-00000076.soc	4	ex38	ED-00015-00000071.soc	4
ex4	ED-00015-00000030.soc	4	ex39	ED-00015-00000046.soc	4
ex5	ED-00006-00000048.soc	9	ex40	ED-00015-00000057.soc	4
ex6	ED-00015-00000031.soc	4	ex41	ED-00015-00000037.soc	4
ex7	ED-00006-00000033.soc	9	ex42	ED-00015-00000078.soc	4
ex8	ED-00015-00000075.soc	4	ex43	ED-00006-00000036.soc	9
ex9	ED-00015-00000047.soc	4	ex44	ED-00015-00000044.soc	4
ex10	ED-00012-00000001.soc	30	ex45	ED-00015-00000026.soc	4
ex11	ED-00015-00000016.soc	4	ex46	ED-00015-00000050.soc	4
ex12	ED-00015-00000070.soc	4	ex47	ED-00015-00000073.soc	4
ex13	ED-00015-00000011.soc	4	ex48	ED-00015-00000041.soc	4
ex14	ED-00006-00000034.soc	9	ex49	ED-00015-00000069.soc	4
ex15	ED-00015-00000064.soc	4	ex50	ED-00015-00000066.soc	4
ex16	ED-00015-00000058.soc	4	ex51	ED-00015-00000039.soc	4
ex17	ED-00006-00000003.soc	9	ex52	ED-00006-00000021.soc	7
ex18	ED-00006-00000008.soc	9	ex53	ED-00015-00000035.soc	4
ex19	ED-00015-00000065.soc	4	ex54	ED-00015-00000062.soc	4
ex20	ED-00015-00000055.soc	4	ex55	ED-00015-00000053.soc	4
ex21	ED-00015-00000072.soc	4	ex56	ED-00015-00000019.soc	4
ex22	ED-00006-00000007.soc	9	ex57	ED-00015-00000045.soc	4
ex23	ED-00015-00000060.soc	4	ex58	ED-00015-00000054.soc	4
ex24	ED-00015-00000074.soc	4	ex59	ED-00015-00000061.soc	4
ex25	ED-00006-00000004.soc	9	ex60	ED-00006-00000032.soc	9
ex26	ED-00015-00000034.soc	4	ex61	ED-00015-00000049.soc	4
ex27	ED-00015-00000051.soc	4	ex62	ED-00006-00000046.soc	7
ex28	ED-00015-00000077.soc	4	ex63	ED-00006-00000037.soc	9
ex29	ED-00015-00000068.soc	4	ex64	ED-00006-00000029.soc	9
ex30	ED-00015-00000010.soc	4	ex65	ED-00006-00000011.soc	9
ex31	ED-00015-00000079.soc	4	ex66	ED-00006-00000022.soc	7
ex32	ED-00006-00000018.soc	9	ex67	ED-00015-00000059.soc	4
ex33	ED-00015-00000043.soc	4	ex68	ED-00015-00000052.soc	4
ex34	ED-00015-00000015.soc	4	ex69	ED-00006-00000012.soc	9
ex35	ED-00006-00000028.soc	9	ex70	ED-00015-00000067.soc	4

TABLEAU 6. IV. Correspondance between the IDs of the examples in Table 6. II and 6. III and their PrefLib.org file name. The column m gives, for each example, its original number of permutations, from which 3 permutations were randomly chosen to compute the data of Table 6. II and 6. III.

		A	B		C	D				A	B		C	D	
ID	n	# 3-cycles	max # 3-cycles	A/B	nb constr.	max nb constr.	C/D	ID	n	# 3-cycles	max # 3-cycles	A/B	nb constr.	max nb constr.	C/D
ex1	29	123	1015	12.1%	260	406	64%	ex36	44	833	3542	23.5%	357	946	37.7%
ex2	20	58	330	17.6%	105	190	55.3%	ex37	18	40	240	16.7%	96	153	62.8%
ex3	44	773	3542	21.8%	430	946	45.5%	ex38	17	25	204	12.3%	94	136	69.1%
ex4	64	2356	10912	21.6%	840	2016	41.7%	ex39	40	443	2660	16.7%	381	780	48.9%
ex5	24	139	572	24.3%	126	276	45.7%	ex40	73	4549	16206	28.1%	768	2628	29.2%
ex6	67	1839	12529	14.7%	1043	2211	47.2%	ex41	67	2844	12529	22.7%	750	2211	33.9%
ex7	23	97	506	19.2%	140	253	55.3%	ex42	12	6	70	8.6%	53	66	80.3%
ex8	42	566	3080	18.4%	402	861	46.7%	ex43	18	29	240	12.1%	106	153	69.3%
ex9	28	145	910	15.9%	219	378	57.9%	ex44	45	954	3795	25.1%	368	990	37.2%
ex10	11	7	55	12.7%	42	55	76.4%	ex45	82	5938	22960	25.9%	1143	3321	34.4%
ex11	70	3331	14280	23.3%	824	2415	34.1%	ex46	26	165	728	22.7%	142	325	43.7%
ex12	67	2846	12529	22.7%	833	2211	37.7%	ex47	36	379	1938	19.6%	307	630	48.7%
ex13	63	1769	10416	17%	908	1953	46.5%	ex48	70	2767	14280	19.4%	966	2415	40%
ex14	23	155	506	30.6%	103	253	40.7%	ex49	81	5358	22140	24.2%	1150	3240	35.5%
ex15	43	566	3311	17.1%	449	903	49.7%	ex50	52	905	5850	15.5%	640	1326	48.3%
ex16	21	100	385	26%	100	210	47.6%	ex51	89	8193	29370	27.9%	1190	3916	30.4%
ex17	14	17	112	15.2%	61	91	67%	ex52	18	72	240	30%	73	153	47.7%
ex18	23	122	506	24.1%	119	253	47%	ex53	68	2575	13090	19.7%	944	2278	41.4%
ex19	40	507	2660	19.1%	374	780	48%	ex54	37	475	2109	22.5%	305	666	45.8%
ex20	52	832	5850	14.2%	689	1326	52%	ex55	23	110	506	21.7%	133	253	52.6%
ex21	17	38	204	18.6%	84	136	61.8%	ex56	87	5744	27434	20.9%	1379	3741	36.9%
ex22	23	85	506	16.8%	155	253	61.3%	ex57	32	296	1360	21.8%	220	496	44.4%
ex23	72	3398	15540	21.9%	1021	2556	40%	ex58	60	1035	8990	11.5%	955	1770	54%
ex24	20	32	330	9.7%	138	190	72.6%	ex59	41	560	2870	19.5%	394	820	48.1%
ex25	14	17	112	15.2%	59	91	64.8%	ex60	23	109	506	21.5%	119	253	47%
ex26	55	2112	6930	30.5%	492	1485	33.1%	ex61	38	453	2280	19.9%	344	703	48.9%
ex27	77	5584	19019	29.4%	805	2926	27.5%	ex62	30	224	1120	20%	206	435	47.4%
ex28	56	1575	7308	21.6%	600	1540	39%	ex63	19	57	285	20%	81	171	47.4%
ex29	32	317	1360	23.3%	228	496	46%	ex64	19	59	285	20.7%	95	171	55.6%
ex30	71	3342	14910	22.4%	889	2485	35.8%	ex65	20	64	330	19.4%	112	190	59%
ex31	41	514	2870	17.9%	397	820	48.4%	ex66	18	31	240	12.9%	101	153	66%
ex32	24	159	572	27.8%	108	276	39.1%	ex67	55	1818	6930	26.2%	533	1485	35.9%
ex33	18	33	240	13.8%	99	153	64.7%	ex68	21	82	385	21.3%	103	210	49.1%
ex34	69	2835	13685	20.7%	984	2346	41.9%	ex69	20	79	330	23.9%	97	190	51.1%
ex35	24	231	572	40.4%	89	276	32.3%	ex70	30	197	1120	17.6%	231	435	53.1%

TABLEAU 6. V. Statistics on randomly uniformly generated data. The size n of each example corresponds to the size of the examples on real data of Table 6. II. For this n, three permutations were randomly and uniformly generated using the Fisher-Yates shuffle [**64**]. Column A gives the number of 3-cycles found in this example, column B gives the maximal number of 3-cycles that could be present in an example of size n (see [**16**] for the formulas for odd and even length cases), column A/B gives the proportion, in %, of 3-cycles found compared to the maximum number of 3-cycles, column C gives the number of constraints $i \prec j$, $1 \leq i \neq j \leq n$ found, column D gives the maximal number of constraints to be found for permutations of size n and finally column C/D gives the % of constraints found.

		A	B		C				A	B		C	
ID	n	time ILP normal	time ILP 3 cyc.thm.	A/B	time 3HS	A/C	ID	n	time ILP normal	time ILP 3 cyc.thm.	A/B	time 3HS	A/C
ex1	29	57.309	29.520	1.9	1.99	28.8	ex36	44	258.971	189.261	1.4	15.877	16.3
ex2	20	18.123	11.007	1.7	0.819	22.1	ex37	18	13.032	6.505	2.0	0.649	20.1
ex3	44	247.767	159.238	1.6	13.751	18.0	ex38	17	10.836	4.743	2.3	0.389	27.9
ex4	64	1187.455	904.222	1.3	141.036	8.4	ex39	40	172.935	107.624	1.6	6.861	25.2
ex5	24	31.999	24.004	1.3	1.940	16.5	ex40	73	10639.121	8172.505	1.3	866.353	12.3
ex6	67	926.143	571.386	1.6	31.726	29.2	ex41	67	1269.947	866.333	1.5	76.922	16.5
ex7	23	27.796	19.054	1.5	1.348	20.6	ex42	12	3.658	1.171	3.1	0.132	27.8
ex8	42	208.341	132.143	1.6	9.284	22.4	ex43	18	13.053	5.759	2.3	0.562	23.2
ex9	28	52.716	30.045	1.8	2.426	21.7	ex44	45	295.712	247.429	1.2	53.682	5.5
ex10	11	2.806	0.942	3.0	0.184	15.3	ex45	82	46019.661	55961.999	0.8	4206.247	10.9
ex11	70	3889.091	1407.340	2.8	415.431	9.4	ex46	26	41.819	30.889	1.4	2.260	18.5
ex12	67	1144.591	929.713	1.2	75.092	15.2	ex47	36	124.236	80.875	1.5	5.575	22.3
ex13	63	883.163	589.764	1.5	48.775	18.1	ex48	70	2657.183	1048.878	2.5	83.453	31.8
ex14	23	28.464	23.320	1.2	2.012	14.2	ex49	81	4548.700	4060.120	1.1	953.759	4.8
ex15	43	403.926	194.758	2.1	17.374	23.3	ex50	52	372.485	244.676	1.5	15.148	24.6
ex16	21	21.823	16.694	1.3	1.411	15.5	ex51	89	702800.102	461614.250	1.5	69214.169	10.2
ex17	14	5.995	2.715	2.2	0.319	18.8	ex52	18	13.507	8.975	1.5	1.057	12.8
ex18	23	28.289	19.712	1.4	1.700	16.6	ex53	68	1124.715	797.670	1.4	66.016	17.0
ex19	40	165.408	104.434	1.6	7.104	23.3	ex54	37	129.996	85.142	1.5	5.832	22.3
ex20	52	747.845	222.887	3.4	24.697	30.3	ex55	23	28.334	19.386	1.5	1.569	18.1
ex21	17	11.039	5.470	2.0	0.637	17.3	ex56	87	4591.207	3709.316	1.2	522.010	8.8
ex22	23	27.823	17.738	1.6	1.204	23.1	ex57	32	80.648	65.864	1.2	4.131	19.5
ex23	72	1863.480	1519.209	1.2	236.237	7.9	ex58	60	792.506	375.007	2.1	160.510	4.9
ex24	20	17.922	7.123	2.5	0.607	29.5	ex59	41	515.166	276.698	1.9	8.181	63.0
ex25	14	5.950	2.626	2.3	0.240	24.8	ex60	23	28.381	20.756	1.4	1.651	17.2
ex26	55	1447.574	580.477	2.5	49.691	29.1	ex61	38	136.756	90.964	1.5	5.884	23.2
ex27	77	4475.162	4385.677	1.0	1059.166	4.2	ex62	30	65.561	44.190	1.5	3.031	21.6
ex28	56	578.007	799.169	0.7	30.233	19.1	ex63	19	15.367	10.678	1.4	0.803	19.1
ex29	32	88.696	60.937	1.5	5.039	17.6	ex64	19	16.054	9.492	1.7	0.915	17.6
ex30	71	1880.580	1539.885	1.2	216.318	8.7	ex65	20	18.177	9.526	1.9	0.995	18.3
ex31	41	306.660	173.185	1.8	7.959	38.5	ex66	18	12.980	6.390	2.0	0.521	25.0
ex32	24	31.513	25.869	1.2	2.249	14.0	ex67	55	645.411	488.770	1.3	36.521	17.7
ex33	18	13.028	6.195	2.1	0.540	24.1	ex68	21	21.029	14.649	1.4	1.214	17.3
ex34	69	1359.295	1082.704	1.3	141.539	9.6	ex69	20	18.476	12.420	1.5	1.291	14.3
ex35	24	39.692	32.967	1.2	3.024	13.1	ex70	30	67.788	38.599	1.8	2.983	22.7

TABLEAU 6. VI. Deterministic (CPLEX's ticks) time to solve each example of Table 6. V. For each example, n represents the length of the permutations, column A gives the time required to solve the Integer Linear Programming (ILP) model of this example, column B gives the time required to solve the ILP model with constraints added from the 3-cycle theorem, column B/A gives the acceleration factor of ILP + 3-cycles with respect to ILP, column C gives the time required to solve the ILP model of the 3-Hitting Set Problem related to this example and finally, column C/A gives the acceleration factor of the 3-Hitting Set Problem with respect to the ILP model.

Chapitre 7

TRAVAUX SUR LE CONSENSUS DE CLASSEMENTS

Ce chapitre présente les travaux effectués sur le problème du consensus de classements qui est la généralisation du problème de la médiane de permutations. Les classements permettent d'avoir des éléments à égalités et d'avoir des éléments absents (Voir Section 1.2). Une revue de littérature sur les classements est présentée à la Section 2.10.

Dans des travaux récents, Pierre Andrieu, un étudiant au doctorat au Laboratoire de Recherche en Informatique à l'université Paris-Sud, a généralisé davantage le score de Kemeny en utilisant le concept d'un schéma de score. Cette notion est brièvement introduite dans la Section 1.3.5 mais détaillée davantage dans le travail de la Section 7. C'est sur cette puissante base théorique que les travaux de ce chapitre s'appuient.

Notez que ces travaux ont été réalisés en grande partie dans le cadre d'un stage de recherche au Laboratoire de Recherche en Informatique à l'Université Paris-Sud rendu possible par une bourse de mobilité France-Québec Globalink de Mitacs.

Rank Aggregation Properties

Résumé

Cet article en préparation est basé sur le travail effectué lors du stage Mitacs au Laboratoire de Recherche en Informatique (Paris-Saclay, France).

L'article se base sur la théorie développée par Pierre Andrieu sur les classements et propose une série de propriétés sur les consensus de classements.

La première propriété permet de déterminer l'ordre relatif dans le consensus entre deux éléments qui sont toujours à égalité. Ensuite, la méthode de compression permet de créer une instance plus petite du problème. Quand cette dernière instance est résolue, la méthode de décompression permet de retrouver une solution pour l'instance originale. La deuxième propriété permet de déterminer si des éléments qui sont souvent à égalité vont être à égalité dans le consensus. La troisième propriété permet de déterminer si un élément va être avant un autre élément dans le consensus.

Notez que nous attendons la publication des travaux de Pierre pour soumettre l'article à une conférence pertinente.

Partage du travail

Dans les contributions de cet article, Robin Milosz a développé les matrices de coût et les propriétés avant-après. Robin Milosz et Adeline Pierrot ont développé les propriétés d'égalité et les méthodes de compressions. Robin Milosz a implémenté et testé ces méthodes et ces propriétés. Tous les auteurs ont rédigé l'article. Sylvie Hamel et Robin Milosz ont fait une restructuration majeure de l'article.

Rank aggregation properties

Robin Milosz and Adeline Pierrot and Sylvie Hamel

LRI - Universite Paris-Sud,
DIRO - Université de Montréal

7.1. Abstract

The rank aggregation problem consists in finding a consensus ranking of a set of rankings. This consensus represent the "closest" ranking to the given set with respect to a chosen score. The following work describes two types of properties of the consensus. The first type reduces the length of the considered rankings and comes from equality properties and compressions. The second type reduces the search space and comes from, what we call, before/after properties.

7.2. Introduction

Given a multiset of rankings, one might be interested in finding a ranking which summarizes the best those rankings. The rank aggregation method intends to find that consensus ranking by searching for the closest ranking to that multiset given a score function. Rankings of this multiset can contain ties between elements and can have some elements missing.

Applications of this problem are numerous. In bioinformatics, rank aggregation can produce a higher quality ranking of genes involved in a genetic disease, by aggregating rankings given by prediction algorithms [**40**]. It is used also in the meta-search engine Conqur-bio [**30**], which reformulates search queries in bioinformatic databases with synonyms keywords and outputs a ranking which summarizes the results of all reformulations. In computer science, rank aggregation can be used to build a meta-search engine which combines the results of different search engines [**56**]. In [**129**], the rank aggregation method is used to predict influencers in social medias based on rankings of users by different criterions.

The case where all rankings of the multiset are containing the same set of elements and have no equalities between elements (therefore permutations), is called the Kemeny-Young method and as been well studied in the past years (see, for example, the two comparative studies [**6**] and [**117**]). The problem is known to be NP-Hard even for a really small

set of four permutations [56]. The advantage of using rankings over permutations is in reflecting better the reality of certain applications where all elements cannot be compared simultaneously or where some elements cannot be distinguished.

An interesting work has been done in [31] which is the first comparison study of rank aggregation algorithms. In this work, many known heuristic and exact approaches for the Kemeny-Young method where adapted to the case of rankings, then compared and tested over different datasets.

Few search space reduction approaches have been developed for the Kemeny-Young method. In [133], a graph based approach called the Extended Condorcet criterion is developed. This approach gives a partial ordering of the elements based on a partition of those elements. In [26], the Pareto criterion is proven. This intuitive criterion states that if an element is always preceding another element in all permutations of a multiset, then any consensus of that multiset should present that order. Authors of [18] propose a method to identify pivot elements that will separate elements in the consensus and therefore break the problem into independent sub-problems. Finally, a pairwise ordering approach is developed in [96].

As the rank aggregation method is more recent than the Kemeny-Young method, no search space reduction work had been done on this subject. Therefore, this paper aims at providing a first search space reduction that will be useful to solve the problem when integrated to any algorithm or heuristic.

7.3. Basic definitions

Let U be the universe of all elements that will be considered in our rankings. Without loss of generality, we identify those elements by integers such as $U = \{1,2,...|U|\}$.

Definition 11. *Let U be the universe of considered elements. A* **ranking** *is an ordered partition of S, $S \subseteq U$. If $S = U$, the ranking is said to be a* **complete** *ranking, if not, it is considered* **incomplete**.

Definition 12. *In a ranking, all the elements that are in a same subset of the partition are said to be in the same* **bucket**. *Thus a ranking r is a list of buckets $b_\ell, \ell \in 1,...,k$ such that $r = [b_1, b_2, ..., b_k]$ and $\cup_{i=1}^{k} b_i = S \subseteq U$.*

Definition 13. *Given a ranking $r = [b_1, b_2, ..., b_k]$, the* **position** *of an element i is $r[i] = \ell$ if $i \in b_\ell$. Note that all the elements in the same bucket have the same position.*

Definition 14. *Let $r = [b_1, b_2, ..., b_k]$ be a ranking on $S \subseteq U$. Let $1 \leq i \neq j \leq |U|$ be two elements of U. If i and j are in S, we define the following relations :*

- i **is tied to** j, *denoted* $i \equiv_r j$, *if* $r[i] = r[j]$ *i.e. if* i *and* j *are in the same bucket.*
- i **is to the left of (or before)** j, *denoted* $i \prec_r j$, *if* $r[i] < r[j]$.
- i **is to the right of (or after)** j, *denoted* $i \succ_r j$, *if* $r[i] > r[j]$.

If either i *or* j *are not in* S, *we have an incomplete ranking and we define three new relations :* $i!j_r$ *(resp.* $!ij_r$*) if the element* i *(resp* j*) is present in* r *but not* j *(resp not* i*) and* $!i!j_r$ *if both elements are absent from the ranking* r. *Note that if there are no ambiguity on the ranking* r, *we can omit it in the notations.*

Observation 2. *For any ranking* r *on* $S \subseteq U$ *and any pair of elements* $1 \leq i \neq j \leq |U|$, *one and only one of the relations* $i \equiv j$, $i \prec j$, $i \succ j$, $i!j$, $!ij$ *and* $!i!j$ *is verified.*

Example 5. *Let* $r = [\{1,2,3\},\{4\},\{7,6\},\{8\}]$ *be a ranking of* $U = \{1,2,\ldots,8\}$. *In* r, *there are 7 elements and 4 buckets. Elements* 1,2 *and* 3 *are in the same bucket* $(1 \equiv 2, 2 \equiv 3, 1 \equiv 3)$. *Element* 1 *is at the left of element* 4 $(1 \prec 4)$. *Element* 4 *is alone in its bucket. The relation between* 8 *and* 5 *is* $8!5$. *The position of element* 7 *is* $r[7] = 3$. *The ranking* r *is incomplete since element* 5 *is missing.*

As a side note, permutations are equivalent to complete rankings that have all buckets of size one.

Though a lot of work have been done on the permutation aggregation problem, the interest of considering incomplete rankings comes from a twofold advantage : first, enabling the use of ties in situations where elements cannot be distinguished from each other, and second, enabling rankings to be incomplete in situations where elements may be missing in some rankings.

In the next section, we formalize the criteria that will define a consensus of a set of rankings.

7.4. Scoring scheme, Kemeny score and cost matrices

The rank aggregation problem problem consists in finding a ranking which is a consensus of a set of rankings. Following (**ref Pierre Andrieu et al.**), we now introduce the scoring scheme and the Kemeny score in order to define this consensus.

Definition 15. *A* **scoring scheme** Q *is a couple of vectors* $Q = (V,V') \in \mathbb{R}^6 \times \mathbb{R}^6$ *representing the penalty of changing the order relation of pairs of elements.*

More precisely,

— $V[1]$ *is the penalty of letting* $i \prec j$ *stay* $i \prec j$,

— $V[2]$ *is the penalty of changing from* $i \succ j$ *to* $i \prec j$,

— $V[3]$ *is the penalty of changing from* $i \equiv j$ *to* $i \prec j$,

— $V[4]$ *is the penalty of changing from* $i!j$ *to* $i \prec j$,
— $V[5]$ *is the penalty of changing from* $!ij$ *to* $i \prec j$,
— $V[6]$ *is the penalty of changing from* $!i!j$ *to* $i \prec j$,

— $V'[1]$ *is the penalty of changing from* $i \prec j$ *to* $i \equiv j$,
— $V'[2]$ *is the penalty of changing from* $i \succ j$ *to* $i \equiv j$,
— $V'[3]$ *is the penalty of letting* $i \equiv j$ *stay* $i \equiv j$,
— $V'[4]$ *is the penalty of changing from* $i!j$ *to* $i \equiv j$,
— $V'[5]$ *is the penalty of changing from* $!ij$ *to* $i \equiv j$,
— $V'[6]$ *is the penalty of changing from* $!i!j$ *to* $i \equiv j$.

Let $\mathcal{R}$ be a multiset of rankings over U. To characterize the ordering of elements in $\mathcal{R}$, we define the vector $\Omega_{ij}^{\mathcal{R}}$ of length 6 for all pairs $i \neq j$, $i,j \in U$ such that :

— $\omega_{i<j}^{\mathcal{R}} = \#\{r \in \mathcal{R} : i \prec j\}$
— $\omega_{i>j}^{\mathcal{R}} = \#\{r \in \mathcal{R} : i \succ j\}$
— $\omega_{i\equiv j}^{\mathcal{R}} = \#\{r \in \mathcal{R} : i \equiv j\}$
— $\omega_{i}^{\mathcal{R}} = \#\{r \in \mathcal{R} : i!j\}$
— $\omega_{j}^{\mathcal{R}} = \#\{r \in \mathcal{R} : !ij\}$
— $\omega_{\varnothing}^{\mathcal{R}} = \#\{r \in \mathcal{R} : !i!j\}$
— $\Omega_{ij}^{\mathcal{R}} = (\omega_{i<j}^{\mathcal{R}}, \omega_{i>j}^{\mathcal{R}}, \omega_{i\equiv j}^{\mathcal{R}}, \omega_{i}^{\mathcal{R}}, \omega_{j}^{\mathcal{R}}, \omega_{\varnothing}^{\mathcal{R}})$

We now have everything we need to define our score and rank aggregation problem.

Definition 16. *Given a multiset of rankings* $\mathcal{R}$ *over* U *and a scoring scheme* $Q = (V,V')$, *the* **generalized Kemeny score** *of a ranking* r *with respect to* $\mathcal{R}$ *is defined as :*

$$K^Q(r,\mathcal{R}) = \sum_{\substack{(i,j)\in U^2,\\ r[i]<r[j]}} |<V,\Omega_{ij}^{\mathcal{R}}>| + \sum_{\substack{(i,j)\in U^2,\\ i<j,\\ r[i]=r[j]}} |<V',\Omega_{ij}^{\mathcal{R}}>|,$$

where $|<X,Y>|$ *is the absolute value of the scalar product of the two vectors* X *and* Y. $K^Q(r,\mathcal{R})$ *is denoted simply* $K(r,\mathcal{R})$ *if there is no ambiguity.*

In other words, we sum for each pair of elements the penalty of having this pair in that order in r compared to the multiset $\mathcal{R}$. Note that when $\mathcal{R}$ is a multiset of permutations, the usual Kemeny score of the literature [76] where we sum Kendall-τ distances [78] between the consensus and the rankings of $\mathcal{R}$ corresponds to the scoring scheme $Q = \big((0,1,0,0,0,0),(0,0,0,0,0,0)\big)$. When $\mathcal{R}$ is a multiset of complete rankings with ties [31], summing the generalized Kendall-τ distance between the consensus and the rankings

of $\mathcal{R}$ can be characterized by the scoring scheme $Q = \big((0,1,p,0,0,0),(p,p,0,0,0,0)\big)$, with $p \in [0,1]$.

Definition 17. *Given a multiset of rankings $\mathcal{R}$ over U and a scoring scheme Q,* **the rank aggregation problem** *consist in finding a complete ranking c over U, called* **consensus** *ranking, that minimise the generalized Kemeny score with respect to $\mathcal{R}$ i.e such that*

$$K(c,\mathcal{R}) \leq K(r,\mathcal{R}), \forall r \text{ complete rankings over } U.$$

Note that rankings in $\mathcal{R}$ can be complete or incomplete but the consensus is a complete ranking. Note also that the consensus is not always unique and that we will define the **consensus set** of $\mathcal{R}$ as the set of all consensus rankings of $\mathcal{R}$.

To simplify computation and notation, we can use cost matrices to compute the generalized Kemeny score.

Definition 18. *Given a multiset of rankings $\mathcal{R}$ over U, the* **left cost matrix** $CL(\mathcal{R})$ *is the matrix where $CL_{ij}(\mathcal{R})$ represent the cost of having i to the left of j in the consensus ranking, i.e $CL_{ij}(\mathcal{R}) = |< V,\Omega_{ij}^{\mathcal{R}} >|$. In the same way, let $CR(\mathcal{R})$ and $CE(\mathcal{R})$) be the* **right cost matrix** *and the* **equality cost matrix**, *respectively representing the costs of having an element at the right of another ($CR_{ij}(\mathcal{R}) = |< V,\Omega_{ji}^{\mathcal{R}} >|$) or tied to it ($CE_{ij}(\mathcal{R}) = |< V',\Omega_{ij}^{\mathcal{R}} >|$).*

Note that by definition, we have $CR_{ji}(\mathcal{R}) = |< V,\Omega_{ij}^{\mathcal{R}} >| = CL_{ij}(\mathcal{R})$. When there is no confusion, we will simply write CL_{ij}, CR_{ij} and CE_{ij} instead of $CL_{ij}(\mathcal{R})$, $CR_{ij}(\mathcal{R})$ and $CE_{ij}(\mathcal{R})$.

Definition 19. *The cost matrices are* **consistent** *if $\forall i \neq j \in U$, $CE_{ij} = CE_{ji}$. A scoring scheme Q is* **consistent** *if the associated vector v' have the properties $V'[1] = V'[2]$ and $V'[4] = V'[5]$.*

Proposition 2. *Let Q be a consistent scoring scheme, then for any ranking multiset $\mathcal{R}$, the cost matrices associated to Q and $\mathcal{R}$ are consistent.*

We consider in the remaining of this work that our scoring schemes and our cost matrices are consistent.

Given the cost matrices, the generalized Kemeny score can be computed as follows :

$$K(r,\mathcal{R}) = \sum_{\substack{(i,j)\in U^2,\\ r[i]<r[j]}} CL_{ij}(\mathcal{R}) + \sum_{\substack{(i,j)\in U^2,\\ i<j,\\ r[i]=r[j]}} CE_{ij}(\mathcal{R}).$$

Example 6. *The generalized Kemeny score between a ranking $r = [\{1\},\{2,3\}]$ and a multiset of rankings $\mathcal{R}$ is $CL_{12}(\mathcal{R})+CL_{13}(\mathcal{R})+CE_{23}(\mathcal{R})$ denoted simply $CL_{12}+CL_{13}+CE_{23}$*

if there is no ambiguity. Note that tthis score can also be expressed with the right cost matrix as $CR_{21} + CR_{31} + CE_{23}$*. If the cost matrices are consistent, the score can also be expressed as* $CL_{12} + CL_{13} + CE_{32}$*.*

Definition 20. *An* **instance** *of the rank aggregation problem is a triplet of cost matrices* (CL, CR, CE) *over* $U \times U$*.*

For each couple ($\mathcal{R}$,Q) we have defined three cost matrices which allow to compute the generalized Kemeny score. For reasons of compactness, implementation and clarity we will only use those cost matrices in the remaining of this work. Thus the forthcoming properties are stated in a way that can be verified using only the cost matrices, regardless of the initial ranking multiset or scoring scheme.

7.5. Equality property and compressions

In this section we propose a first equality property which is based on the intuitive idea that two elements that are together in the same bucket in each ranking of $\mathcal{R}$ should be in the same bucket in a consensus of $\mathcal{R}$. Following this, we show a method that reduces the size of an instance based on the equality property, making the problem easier to solve.

Theorem 5. *Let* $\mathcal{R}$ *be a multiset of rankings over* U*. Let* CL*,* CR *and* CE *be the cost matrices of* $\mathcal{R}$ *with respect to a consistent scoring scheme* Q*. Let* $i \neq j$ *be two elements of* U*. If* $i \equiv j$ *in all rankings of* $\mathcal{R}$ *and* $CE_{ij} \leq \min\{CL_{ij}, CR_{ij}\}$ *(resp.* $CE_{ij} < \min\{CL_{ij}, CR_{ij}\}$*) then* $i \equiv j$ *in at least one (resp. in all) consensus of* $\mathcal{R}$*.*

Proof. First note that if $i \equiv j$ in all rankings of $\mathcal{R}$, then we have that

$$\forall k \neq i \neq j, CL_{ik} = CL_{jk}, CR_{ik} = CR_{jk} \text{ and } CE_{ik} = CE_{jk}. \qquad (\diamond)$$

Now, let us assume there is a consensus c of $\mathcal{R}$ in which elements i and j are not tied (i.e $c[i] \neq c[j]$). We will show that putting elements i and j in the same bucket will either reduce the generalized Kemeny score, thus contradicting the fact that c is a consensus, or keep the same score, thus giving us a consensus with the desired property.

Case 1. $i \prec_c j$:

Let K be the part of c consisting of the elements which have a position strictly between i and j, L be the set of elements that are tied with i in c and M be the set of elements that are tied with j in c. Then $c = [..., \{i \cup L\}, K, \{j \cup M\}, ...]$. When computing the generalized Kemeny score, if we put i and j at the same position, only terms involving pairs containing i or j and an element of $L \cup K \cup M$ $L \cup K \cup M \cup \{i,j\}$ will change value.

Let c' (resp. c'') be the new ranking obtained from c by putting i (resp. j) in the same bucket of j (resp. i). So we have, $c' = [...,\{L\},K,\{i,j \cup M\},...]$ and $c'' = [...,\{i,j \cup L\},K,\{M\},...]$. We will now show that either c' or c'' is a better consensus than c, contradicting our hypothesis.
We will consider two subcases :

Case A) $\sum_{l\in L} CR_{jl} - CE_{il} + \sum_{k\in K} CR_{jk} - CL_{ik} + \sum_{m\in M} CE_{jm} - CL_{im} < 0$.

In that case, we calculate the difference of generalized Kemeny score, $\Delta = K(c',\mathcal{R}) - K(c,\mathcal{R})$, between c' and c :

$$\begin{aligned}
\Delta &\overset{(1)}{=} CE_{ij} - CL_{ij} + \sum_{l\in L} CR_{il} - CE_{il} + \sum_{k\in K} CR_{ik} - CL_{ik} + \sum_{m\in M} CE_{im} - CL_{im} \\
&\overset{(\diamond)}{=} CE_{ij} - CL_{ij} + \sum_{l\in L} CR_{jl} - CE_{il} + \sum_{k\in K} CR_{jk} - CL_{ik} + \sum_{m\in M} CE_{jm} - CL_{im} \\
&\overset{(2)}{<} CE_{ij} - CL_{ij} \\
&\overset{(3)}{\leq} 0
\end{aligned}$$

Case B) $\sum_{l\in L} CR_{jl} - CE_{il} + \sum_{k\in K} CR_{jk} - CL_{ik} + \sum_{m\in M} CE_{jm} - CL_{im} \geq 0$.

In that case, we calculate the difference of generalized Kemeny score, $\Delta = K(c'',\mathcal{R}) - K(c,\mathcal{R})$, between c'' and c :

$$\begin{aligned}
\Delta &\overset{(1)}{=} CE_{ij} - CL_{ij} + \sum_{l\in L} CE_{jl} - CR_{jl} + \sum_{k\in K} CL_{jk} - CR_{jk} + \sum_{m\in M} CL_{jm} - CE_{jm} \\
&\overset{(\diamond)}{=} CE_{ij} - CL_{ij} + \sum_{l\in L} CE_{il} - CR_{jl} + \sum_{k\in K} CL_{ik} - CR_{jk} + \sum_{m\in M} CL_{im} - CE_{jm} \\
&\overset{(2)}{\leq} CE_{ij} - CL_{ij} \\
&\overset{(3)}{\leq} 0
\end{aligned}$$

In both subcases, (1) comes from considering the contribution of pairs whose order was changed, (2) comes from the conditions of each subcases and (3) from the statement of the theorem. In all cases, we either lower the generalized Kemeny score or keep it the same, showing the existence of a consensus with i and j in the same bucket ($i \equiv j$). Moreover if $CE_{ij} < min\{CL_{ij},CR_{ij}\}$ then the generalized Kemeny score is strictly reduced, contradicting the existence of a consensus c such that $i \not\equiv j$.

Case 2. $j \prec_c i$: Since Q is a consistent score scheme, this case is symmetric to the preceding one. ■

Corollary 7.5.1. *Let $\mathcal{R}$ be a multiset of rankings over U. Let CL, CR and CE be the cost matrices of $\mathcal{R}$ with respect to a consistent scoring scheme Q. If $CE_{ij} = 0$ and $\forall k \in U, k \neq i,j, CL_{ik} = CL_{jk}, CR_{ik} = CR_{jk}, CE_{ik} = CE_{jk}$ then $i \equiv j$ in at least one consensus of $\mathcal{R}$.*

Proof. By definition of the cost matrices, $min\{CL_{ij},CR_{ij}\} \geq 0 = CE_{ij}$ and we apply Theorem 5. ∎

We describe now a compression method to reduce the size of the problem. Elements that are proven to be tied for any consensus of $\mathcal{R}$ can be compressed into heavier elements and handled as a whole as they will be in the same bucket in a consensus.

Definition 21. *Let $\mathcal{R}$ be a multiset of rankings over U. Let $U' \subset U$. A* **compression** *of U with respect to $\mathcal{R}$ is a surjective function $C_{\mathcal{R}} : U \longrightarrow U'$ such that $C_{\mathcal{R}}(i) = C_{\mathcal{R}}(j) \Longleftrightarrow i \equiv j$ in all consensus of $\mathcal{R}$. U' is called the set of compressed elements.*

By extension, we define the **compression of an instance** *as the function that takes the three cost matrices from the original instance as input and outputs three cost matrices of the compressed instance such that $\forall i' \neq j' \in U'$,*

$$CL'_{i'j'} = \sum_{\substack{i \neq j \in U \\ C_{\mathcal{R}}[i]=i' \\ C_{\mathcal{R}}[j]=j'}} CL_{ij}, \quad CR'_{i'j'} = \sum_{\substack{i \neq j \in U \\ C_{\mathcal{R}}[i]=i' \\ C_{\mathcal{R}}[j]=j'}} CR_{ij}, \quad CE'_{i'j'} = \sum_{\substack{i \neq j \in U \\ C_{\mathcal{R}}[i]=i' \\ C_{\mathcal{R}}[j]=j'}} CE_{ij}.$$

This gives a compressed instance that is equivalent to the original instance in terms of the generalized Kemeny score of a consensus.

After finding a consensus for a compressed instance of the problem, we can decompress the result to obtain a consensus for the original instance by enumerating for each heavy element in the result its corresponding set of elements.

Definition 22. *Let $\mathcal{R}$ be a multiset of rankings over U, $C_{\mathcal{R}}$ be a compression of U with respect to $\mathcal{R}$, U' be the set of compressed elements and $\mathcal{P}(U)$ be the the set of all subsets of U. A* **decompression** *of U' with respect to $\mathcal{R}$ is a function $D_{C_{\mathcal{R}}} : U' \longrightarrow \mathcal{P}(U)$ such that $\forall i' \in U', D_{C_{\mathcal{R}}}(i') = \{i \in U \,|\, C_{\mathcal{R}}(i) = i'\}$.*

By extension, we define the **decompression of a ranking**, *associated to a compression $C_{\mathcal{R}}$, as a function that takes a ranking r' over U' and outputs a ranking r over U such that $r[i] = r'[C_{\mathcal{R}}[i]]$, $\forall i \in U$.*

The following theorem follows easily from the definitions of compression and decompression and Theorem 5.

Theorem 6. *Let $\mathcal{R}$ be a multiset of rankings over U. Let CL, CR and CE be the consistent cost matrices of an instance and let $C_{\mathcal{R}}$ be a compression. Let c' be a consensus ranking of the compressed instance (CL', CR' and CE'). Then $D_{C_{\mathcal{R}}}(c')$ is a consensus of the original instance (CL, CR and CE).*

Compressions are of great importance as they reduce the size of the problem and avoid unnecessary handling of repeated elements. It is also of great use in some local search heuristics [40] where elements are individually moved from one bucket to the other. In those heuristics, moving one by one an entire bucket of tied elements may be prohibited by having greater Kemeny scores in intermediate rankings, while moving the corresponding heavy element may be advantageous.

As the equality property can reduce the search space and even the size of the instance through compression methods, its application may not occur very often as two elements need to always share the same position in all rankings. It is then useful to investiguate cases where two elements are often together. The next section will inspect thoses cases in detail.

7.6. Bounded-interference properties

In this section, we introduce the bounded-interference properties for equality and before or after properties between pairs of elements. This approach is a generalization of the Major Order theorems of [96]. The principle behind those properties is that when a pair or a subset of elements share a common relative order in most of the rankings and are positioned not too far away within the other rankings (where they do not share relative orders), then this pair or subset shall keep this order in the consensus ranking.

7.6.1. Equality property

The next property is based on the idea that if two elements are often in the same bucket in rankings of $\mathcal{R}$ and close to each other in the other rankings, then they might be together in a consensus ranking. It extends the previous equality property. To derive a theorem for this property, we first need to define the interference set of a pair of elements.

Definition 23. *The* **interference set of a pair of elements** *i,j, $i \neq j \in U$ with respect to a set of rankings $\mathcal{R}$, denoted $Y_{ij}^{\mathcal{R}} \subseteq U \setminus \{i,j\}$, is the set of elements that may interfere in between i and j i.e. changing the order between i and j in a ranking $r \in \mathcal{R}$ may also change the order between i,j and elements of $Y_{ij}^{\mathcal{R}}$ in r. If an element has been shown to*

precede or succeed both i and j in any consensus c of $\mathcal{R}$ it cannot interfere. Formally :

$$Y_{ij}^{\mathcal{R}} = \{y \in U \setminus \{i,j\} \ : \ \neg(y \prec_c i \wedge y \prec_c j) \ \wedge \ \neg(y \succ_c i \wedge y \succ_c j)\},$$

where c is any consensus of $\mathcal{R}$.

Note that by definition, we have $Y_{ij}^{\mathcal{R}} = Y_{ji}^{\mathcal{R}}$. Again, when there is no confusion, we will simply write Y_{ij} instead of $Y_{ij}^{\mathcal{R}}$.

Theorem 7. *Let $\mathcal{R}$ be a multiset of rankings over U. Let CL, CR and CE be the cost matrices of $\mathcal{R}$ with respect to a consistent scoring scheme Q. Let Y_{ij} be the interfering set for pair (i,j). If*

$$(-CL_{ij} + CE_{ij}) + \sum_{y \in Y_{ij}} max\{CR_{iy} - CR_{jy}, CE_{iy} - CE_{jy}, 0\} < 0,$$

and

$$(-CL_{ij} + CE_{ij}) + \sum_{y \in Y_{ij}} max\{CE_{jy} - CE_{iy}, CL_{jy} - CL_{iy}, 0\} < 0,$$

then $i \not\prec_c j$ (i cannot precede j), where c is any consensus of $\mathcal{R}$.

Proof. Assume there is a consensus c of $\mathcal{R}$ such that $i \prec j$. As in the proof of Theorem 5, we will show that putting elements i and j in the same bucket will reduce the generalized Kemeny score, thus contradicting the fact that c is a consensus.

Let K be the part of c consisting of the elements which have a position strictly between i and j, L be the set of elements that are tied with i in c and M be the set of elements that are tied with j in c. Then $c = [..., \{i \cup L\}, K, \{j \cup M\}, ...]$. When computing the generalized Kemeny score, if we put i and j at the same position, only terms involving pairs containing i or j and an element of $L \cup K \cup M$ will change value. Note that elements $l \in L, k \in K$ and $m \in M$ are all in the interference set Y_{ij}.

Let c' (resp. c'') be the new ranking obtained from c by putting i (resp. j) in the same bucket of j (resp. i). So we have, $c' = [..., \{L\}, K, \{i, j \cup M\}, ...]$ and $c'' = [..., \{i, j \cup L\}, K, \{M\}, ...]$. We will now show that either c' or c'' is a better consensus than c, contradicting our hypothesis.

We will consider two cases :

Case 1) $\sum_{l \in L} CR_{jl} + \sum_{k \in K} CR_{jk} + \sum_{m \in M} CE_{jm} < \sum_{l \in L} CE_{il} + \sum_{k \in K} CL_{ik} + \sum_{m \in M} CL_{im}$.

In that case, we calculate the difference of generalized Kemeny score, $\Delta = K(c', \mathcal{R}) - K(c, \mathcal{R})$, between c' and c :

$$\begin{aligned}\Delta &\overset{(1)}{=} -(\sum_{l\in L} CE_{il} + \sum_{k\in K} CL_{ik} + \sum_{m\in M} CL_{im}) \\ &\quad +(\sum_{l\in L} CR_{il} + \sum_{k\in K} CR_{ik} + \sum_{m\in M} CE_{im}) - CL_{ij} + CE_{ij} \\ &= -(\sum_{l\in L} CE_{il} + \sum_{k\in K} CL_{ik} + \sum_{m\in M} CL_{im}) \\ &\quad +(\sum_{l\in L} CR_{jl} + \sum_{k\in K} CR_{jk} + \sum_{m\in M} CE_{jm}) + p - CL_{ij} + CE_{ij}\end{aligned}$$

Where penalty $p = \sum_{l\in L}(CR_{il} - CR_{jl}) + \sum_{k\in K}(CR_{ik} - CR_{jk}) + \sum_{m\in M}(CE_{im} - CE_{jm})$. Then,

$$\begin{aligned}\Delta &\overset{(2)}{<} p - CL_{ij} + CE_{ij} \\ &\overset{(3)}{\leq} \textstyle\sum_{y\in Y_{ij}} max\{CR_{iy} - CR_{jy}, CE_{iy} - CE_{jy}, 0\} - CL_{ij} + CE_{ij} \\ &\overset{(4)}{<} 0.\end{aligned}$$

Case 2) $\sum_{l\in L} CR_{jl} + \sum_{k\in K} CR_{jk} + \sum_{m\in M} CE_{jm} \geq \sum_{l\in L} CE_{il} + \sum_{k\in K} CL_{ik} + \sum_{m\in M} CL_{im}$.
In that case, we calculate the difference of generalized Kemeny score, $\Delta = K(c'',\mathcal{R}) - K(c,\mathcal{R})$ between c'' and c :

$$\begin{aligned}\Delta &\overset{(1)}{=} -(\sum_{l\in L} CR_{jl} + \sum_{k\in K} CR_{jk} + \sum_{m\in M} CE_{jm}) \\ &\quad +(\sum_{l\in L} CE_{jl} + \sum_{k\in K} CL_{jk} + \sum_{m\in M} CL_{jm}) - CL_{ij} + CE_{ij} \\ &= -(\sum_{l\in L} CR_{jl} + \sum_{k\in K} CR_{jk} + \sum_{m\in M} CE_{jm}) \\ &\quad +(\sum_{l\in L} CE_{il} + \sum_{k\in K} CL_{ik} + \sum_{m\in M} CL_{im}) + p - CL_{ij} + CE_{ij}\end{aligned}$$

Where penalty $p = \sum_{l\in L}(CE_{jl} - CE_{il}) + \sum_{k\in K}(CL_{jk} - CL_{ik}) + \sum_{m\in M}(CL_{jm} - CL_{im})$. Then,

$$\begin{aligned}&\overset{(2)}{\leq} p - CL_{ij} + CE_{ij} \\ &\overset{(3)}{\leq} \textstyle\sum_{y\in Y_{ij}} max\{CE_{jy} - CE_{iy}, CL_{jy} - CL_{iy}, 0\} - CL_{ij} + CE_{ij} \\ &\overset{(4)}{<} 0.\end{aligned}$$

In both cases, (1) comes from considering the contribution of pairs whose order was changed, (2) comes from the special condition of each cases, (3)comes from the fact that p is bounded by the sum (the worst penalty possible) and (4) comes from the statement of the theorem.

In both cases we obtain a new ranking with a lower generalized Kemeny score contradicting our supposed consensus that had $i \prec j$. Thus $i \not\prec j$ in all consensus of $\mathcal{R}$. ■

Lemma 1. *Let $\mathcal{R}$ be a multiset of rankings. If Theorem 7 holds for pair (i,j) and pair (j,i) then $i \equiv j$ for any consensus c of $\mathcal{R}$.*

Proof. Trivially, suppose a consensus c with $c[i] \neq c[j]$, then either $i \prec j$ or $j \prec i$. Each case contradicts the premise. Then $i \equiv j$ in every consensus. ■

After having verified Theorem 1 on all pairs, we can take the transitive closure on all the pairs such that $i \equiv j$ and $j \equiv k$ to get $i \equiv k$.

We can then use the same compression method as described in Section 7.5 but with a carefully added detail. Because CE_{ij} may not be equal to zero for certain pairs such that $i \equiv j$, we have to keep track of the compression cost C_{cost} :

$$C_{cost} = \frac{1}{2} \sum_{i,j \in U \,:\, i \equiv j} CE_{ij}$$

This compression cost will be added to the generalized Kemeny score of a consensus ranking of the compressed instance.

7.6.2. Before property

The following property takes roots in the idea that if an element i is before element j in most rankings of $\mathcal{R}$, and close to it in the remaining rankings, then it should be in that order in a consensus of $\mathcal{R}$.

In the following we quantify the worst possible interference that other elements may give towards not having i before j. More precisely, for each element $y \in Y_{ij}$, we are going to add up its worst interference in a ranking where i is not before j.

Theorem 8. *Let $\mathcal{R}$ be a multiset of rankings over U. Let CL, CR and CE be the cost matrices of $\mathcal{R}$ with respect to a consistent scoring scheme Q. Let Y_{ij} be the interfering set for pair (i,j).*

If

$$(-CE_{ij} + CL_{ij}) + \sum_{y \in Y_{ij}} max\{CL_{iy} - CE_{iy} + CR_{jy} - CE_{jy}, 0\} < 0,$$

and

$$(-CR_{ij} + CL_{ij}) + \sum_{y \in Y_{ij}} max\{CL_{iy} - CE_{jy} + CR_{jy} - CR_{iy}, \\ CL_{iy} - CR_{iy} + CR_{jy} - CL_{jy}, CL_{iy} - CE_{iy} + CR_{jy} - CL_{jy}, 0\} < 0,$$

then $i \prec_c j$, where c is any consensus of $\mathcal{R}$.

Proof. Assume there is a consensus c of $\mathcal{R}$ such that $i \not\prec_c j$. As in the proof of Theorem 5, we will show that putting element i before element j in c will reduce the generalized

Kemeny score, thus contradicting the fact that c is a consensus. We will study the two cases where $i \not\prec_c j$.

Case 1) $j \prec_c i$:
In that case, let K be the part of c consisting of the elements which have a position strictly between i and j, L be the set of elements that are tied with j and M be the set of elements that are tied with i. Then $c = [..., \{j \cup L\}, K, \{i \cup M\}, ...]$. Note that elements $l \in L$,$k \in K$ and $m \in M$ are all in Y_{ij}.

We show that the ranking $c' = [...\{i\},\{L\},K,\{M\},\{j\}...]$ have a smaller generalized Kemeny score, contradicting our choice of consensus.

Let $\Delta = K(c',\mathcal{R}) - K(c,\mathcal{R})$ be the difference of generalized Kemeny score between c' and c. We have :

$$
\begin{aligned}
\Delta \overset{(1)}{=} & -CR_{ij} + CL_{ij} \\
& - \sum_{l \in L} CR_{il} - \sum_{k \in K} CR_{ik} - \sum_{m \in M} CE_{im} + \sum_{l \in L} CL_{il} + \sum_{k \in K} CL_{ik} + \sum_{m \in M} CL_{im} \\
& - \sum_{l \in L} CE_{jl} - \sum_{k \in K} CL_{jk} - \sum_{m \in M} CL_{jm} + \sum_{l \in L} CR_{jl} + \sum_{k \in K} CR_{jk} + \sum_{m \in M} CR_{jm} \\
= & -CR_{ij} + CL_{ij} + \\
& \sum_{l \in L} (CL_{il} - CR_{il} + CR_{jl} - CE_{jl}) + \\
& \sum_{k \in K} (CL_{ik} - CR_{ik} + CR_{jk} - CL_{jk}) + \\
& \sum_{m \in M} (CL_{im} - CE_{im} + CR_{jm} - CL_{jm}) \\
\overset{(2)}{\leq} & -CR_{ij} + CL_{ij} \\
& + \textstyle\sum_{y \in Y_{ij}} max\{CL_{iy} - CE_{jy} + CR_{jy} - CR_{iy}, \\
& CL_{iy} - CR_{iy} + CR_{jy} - CL_{jy}, \\
& CL_{iy} - CE_{iy} + CR_{jy} - CL_{jy}, 0\} \\
\overset{(3)}{<} & 0.
\end{aligned}
$$

Case 2) $i \equiv_c j$:

In that case, let K be the part of c consisting of the elements that are tied with j and let our consensus $c = [...\{i,j \cup K\}...]$.

We show that the ranking $c' = [...\{i\},\{K\},\{j\}...]$ have a smaller generalized Kemeny score, contradicting our choice of consensus.

Let $\Delta = K(c',\mathcal{R}) - K(c,\mathcal{R})$ be the difference of generalized Kemeny score between c' and c. We have :

$$\begin{aligned}
\Delta &\overset{(1)}{=} -CE_{ij} + CL_{ij} + \sum_{k \in K} (CL_{ik} - CR_{ik} + CR_{jk} - CL_{jk}) \\
&\overset{(2)}{\leq} -CR_{ij} + CL_{ij} + \textstyle\sum_{y \in Y_{ij}} max\{CL_{iy} - CR_{iy} + CR_{jy} - CL_{jy}, 0\} \\
&\overset{(3)}{<} 0.
\end{aligned}$$

In both cases, (1) comes from considering the contribution of pairs whose order was changed, (2) comes from the fact that elements $l \in L, k \in K$ and $m \in M$ are all in Y_{ij} and cannot add more to the Kemeny score than being in their most interfering position towards i and j and (3) comes from the statement of the theorem.

In both cases we obtain a new ranking with a lower generalized Kemeny score contradicting our supposed consensus. Thus $i \prec j$ in all consensus of $\mathcal{R}$. ■

7.6.3. Extra note

As an extra note, as new properties are found for an instance of the problem, the interference set for pairs of elements may be reduced, hence potentially enabling more properties to be found. We can iteratively verify properties and compress until no new property can be found. In some preliminary testing, we observed that our space reduction technique iterates only a few times before not beeing able to find new constraints for an instance. Usually the number of iterations was much smaller than the number of elements and the first iterations usually yield the most constraints. Another improvement that can be applied when solving, is to detect when the properties are partitioning elements into subsets that are totally ordered between themselves, following the approach in [**133**]. When this happens, solving the problem can be parallelized by solving each subset and concatenating all solutions in the order of subsets. In some preliminary testing, we observed that this approach can drastically speed up the solving time.

7.7. Conclusion

The rank aggregation problem consist to find a consensus given a set of rankings. This problem has many applications such as building meta search engines. Its simpler version, the Kemeny-Young method, consists of finding a consensus over a set of permutations and has been very well studied in literature. The rank aggregation approach permits to consider ties between elements in rankings and missing elements in rankings. This reflects better the reality of many applications.

In this work, we introduce the problem with a solid theoritical base than can encompass different scenarios in real life applications. As not much work has been done on space reduction technique for this problem, we introduce a few properties that can fix the order of pairs of elements in a consensus ranking. Our first property simply comes from the idea that a pair of elements that are always tied together in all rankings should remain that way in the consensus. We propose a compression method based on that property that creates an equivalent instance of the problem reduced in size and so, easier to solve. Finally, we present the bounded-interference properties that are generalizing the approach of [**96**] that was done for the Kemeny-Young method. Those properties also fix the order of pairs of elements but are more intricate as they tackle pairs of elements that do not always share the same relative order in the set of rankings. All our properties can be integreted into any solver or heuristic for this problem giving a helpful space reduction.

Ideas for further works include testing extensively our space reduction method on synthetic data sets and real life data sets and combining this approch to design efficient heuristics and exact solvers for this problem. Finally, as a lot of work has been done on the permutation case, it would also be worth investigating what knowlegde or approches can be easily transfered to the ranking case.

Aggregate your Rankings with CoRankCo

Résumé

Cet article de conférence présente CoRankCo, une plateforme web sur laquelle on peut calculer des consensus de classements et sera soumis prochainement dans une conférence de base de données (ex. VLDB). Dans cet article, je n'ai fait qu'une petite contribution réalisé dans le cadre du stage Mitacs au Laboratoire de Recherche en Informatique.

La plateforme web CoRankCo (Voir https://corankco.lri.fr/) intègre la base théorique établie par Pierre Andrieu sur les classements et propose plusieurs heuristiques et algorithmes pour calculer le consensus d'un ensemble de classements donné par l'utilisateur. Le schéma de score peut être modifié en fonction des besoins des clients. De plus, une grande banque de données réelles et synthétiques est disponible pour tester les algorithmes.

À notre connaissance, c'est la première plateforme à proposer ces fonctionnalités.

Partage du travail

Dans les contributions de cet article, Robin Milosz a seulement proposé une nouvelle version du modèle en programmation en nombre entiers et il l'a implémenté en Python en utilisant CPLEX. Tout le reste a été développé, implémenté, testé et rédigé par Pierre Andrieu, Bryan Brancotte, Sarah Cohen-Boulakia, Alain Denise, Adeline Pierrot et Stéphane Vialette.

L'article est en préparation et ne peux pas être présenté avant la pulication.

CONCLUSION

Le problème de la médiane de permutations consiste à trouver un consensus d'un ensemble de permutations i.e. une permutation qui minimise la somme des distances aux permutations de l'ensemble. La distance utilisée dans ce travail, la distance de Kendall-τ, compte le nombre de paires d'éléments de ces permutations en désaccord sur leur ordre.

Dans cette thèse, le problème est introduit formellement avec les notations et définitions nécessaires. On introduit aussi le problème généralisé aux classements pouvant contenir des égalités et des éléments absents. Cela est suivi d'une revue de littérature pertinente et complète du sujet et d'un chapitre dédié au contexte et applications. Sont ensuite présentés les travaux réalisés au courant de mon doctorat. Les travaux se concentrent principalement sur la réduction d'espace qui n'a pas été beaucoup explorée dans la littérature. Les autres aspects investigués sont les heuristiques, les bornes inférieures et les approches exactes.

Le premier travail est la suite des travaux de recherche de ma maîtrise et consiste en une version étendue (Journal of Discrete Applied Mathematics) d'un article publié à CALDAM2016. Cette version étend le Major Order Theorem, une méthode de réduction d'espace, à un cas limite applicable à seulement un sous-ensemble de permutations médianes. Une implémentation vectorielle qui permet de calculer très rapidement cette réduction d'espace est expliquée en détails.

Le second travail (présenté à la conférence IWOCA2017) propose trois approches au problème de la médiane de permutations : une heuristique de recuit simulé bien paramétrée, une réduction d'espace de recherche basée sur une combinaison d'approches précédentes et un algorithme exact branch-and-bound raffiné utilisant les deux approches précédentes. Une version étendue de ce travail se trouve en annexe et intègre l'approche par programmation en nombres entiers qui utilise aussi l'heuristique et la réduction d'espace pour accélérer les calculs. Cette version a été acceptée au Journal of Discrete Algorithms en octobre 2018.

Dans un troisième travail (présenté à la conférence IWOCA2018), qui cible directement le cas de la médiane de trois permutations, est présenté une réduction d'espace encore

plus poussée. Celle-ci est basée sur les 3-cycles trouvés dans le graphe de tournoi associé à l'ensemble des permutations $\mathcal{A}$: si une paire d'éléments ne se trouve dans aucun 3-cycle alors elle garde son ordre de majorité dans les médianes de l'ensemble $\mathcal{A}$. Cette réduction d'espace accélère considérablement les approches exactes et généralise toutes les réductions d'espaces connues pour le cas de trois permutations. Nous conjecturons que cette réduction d'espace peut être généralisée aux cas où la cardinalité de l'ensemble de permutations $\mathcal{A}$ est impair. Finalement, un lien intéressant avec le 3-Hitting Set Problem est proposé. Ce lien donne une borne inférieure très stricte au Kemeny Score Problem et nous amène à conjecturer que cette borne inférieure donne en fait le score de Kemeny d'une médiane d'un ensemble de trois permutations.

Le quatrième travail en préparation se concentre sur les propriétés mathématiques et la réduction d'espace pour le problème généralisé des classements. Une méthode de compression permet de réduire la taille de certaines instances pour accélérer autant les heuristiques que les méthodes exactes.

Plusieurs questions restent ouvertes suite à cette thèse. Pour le cas spécial de 3 permutations, dont la complexité demeure inconnue, nous avons posé une conjecture qui dit que résoudre ce problème revient à résoudre le problème du 3-Hitting Set. Nous avons aussi conjecturé que le théorème des 3-cycles est valide pour les ensembles avec un nombre impair de permutations. Celui-ci énonce que si une paire d'éléments ne se trouve dans aucun cycle de longueur 3 dans le graphe de majorité, alors cette paire sera ordonnée selon la majorité des votes dans toutes les médianes. Ces deux conjectures sont importantes car elles permettent d'accélérer énormément le calcul des médianes.

D'autres voies restent d'intérêt et mériteraient d'être explorées. Travailler sur les ensembles de grandes permutations, pouvant contenir des centaines d'éléments, est un succès très récent dans la littérature. Il reste donc place à l'amélioration. Entre autres, les méta-heuristiques de *Variable Neighborhood Search* et de *Noising* semblent bien performer dans des tests préliminaires. Les versions du problème de la médiane dans lesquelles on rajoute ou supprime un élément ou une permutation n'ont pas été investigués et mériteraient un peu de travail. La dernière étude comparative d'heuristiques et d'algorithmes a été faite en 2012, et depuis, plusieurs nouvelles méthodes performantes ont été publiées. Il serait intéressant pour la communauté d'avoir un nouveau travail de comparaison qui inclus ces récents développements. Finalement, la réduction d'espace, les propriétés mathématiques et les classes spéciales du problème sont des approches qui n'ont pas reçues beaucoup

d'attention mais sont bien utiles pour mieux comprendre le problème et pour accélérer les calculs des autres méthodes.

Références

[1] A. Agarwal, H. Raghavan, K. Subbian, P. Melville, R.D. Lawrence, D.C. Gondek et J. Fan : Learning to rank for robust question answering. *In Proceedings of the 21st ACM International Conference on Information and Knowledge Management (CIKM '12)*, pages 833–842, 2012.

[2] N. Ailon : Aggregation of partiel rankings, p-ratings and top-m lists. *Algorithmica*, 57(2):284–300, 2010.

[3] N. Ailon, M. Charikar et N. Newman : Aggregating inconsistent information : ranking and clustering. *Journal of the ACM*, 55(5):1–27, 2008.

[4] J.A. Aledo, J.A. Gámez et D. Molina : Tackling the rank aggregation problem with evolutionary algorithms. *Applied Mathematics and Computation*, 222:632 – 644, 2013.

[5] J.A. Aledo, J.A. Gámez et A. Rosete : Partial evaluation in rank aggregation problems. *Computers and Operations Research*, 78:299 – 304, 2017.

[6] A. Ali et M. Meila : Experiments with kemeny ranking : What works when ? *Mathematical Social Sciences*, 64(1):28–40, 2012.

[7] A.R. Amar : Choosing representatives by lottery voting. *The Yale Law Journal*, 93(7):1283–1308, 1984.

[8] S. Amodio, A. D'Ambrosio et R. Siciliano : Accurate algorithms for identifying the median ranking when dealing with weak and partial rankings under the kemeny axiomatic approach. *European Journal of Operational Research*, 249(2):667 – 676, 2016.

[9] K.J. Arrow : *Social Choice and Individual Values*. Yale University Press, 2nd edition, 144 pages, 1970.

[10] J. Azé, T. Bourquard, S. Hamel, A. Poupon et D.W. Ritchie : Using kendall-τ metabagging to improve protein-protein docking predictions. *In Pattern Recognition in Bioinformatics*, pages 284–295, 2011.

[11] G. Bachmeier, F. Brandt, C. Geist, P. Harrenstein, K. Kardel, D. Peters et H.G. Seedig : k-majority digraphs and the hardness of voting with a constant number of voters. *arXiv : http ://arxiv.org/abs/1704.06304v1*, 45 pages, 2017.

[12] P.S. Badal et A. Das : Efficient algorithms using subiterative convergence for kemeny ranking problem. *Computers & Operations Research*, 98:198–210, 2018.

[13] S. Barberá : Strategy-proofness and pivotal voters : A direct proof of the gibbard-satterthwaite theorem. *International Economic Review*, 24(2):413–417, 1983.

[14] J. Bartholdi III, C.A. Tovey et M.A. Trick : Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare*, 6(2):157–165, 1989.

[15] J.P. Baskin et S. Krishnamurthi : Preference aggregation in group recommender systems for committee decision-making. *In Proceedings of the Third ACM Conference on Recommender Systems (RecSys '09)*, pages 337–340, 2009.

[16] L.W. Beineke et F. Harary : The maximum number of strongly connected subtournaments. *Canadian Mathematical Bulletin*, 8:491–498, 1965.

[17] N. Betzler, R. Bredereck et R. Niedermeier : Partial kernelization for rank aggregation : theory and experiments. *In Proceedings of the 5th International Symposium on Parameterized and Exact Computation, LNCS 6478*, pages 26–37, 2010.

[18] N. Betzler, R. Bredereck et R. Niedermeier : Theoretical and empirical evaluation of data reduction for exact kemeny rank aggregation. *Autonomous Agents and Multi- Agent Systems*, 28(1):721–748, 2014.

[19] N. Betzler, M. R. Fellows, J. Guo, R. Niedermeier et F. A. Rosamond : Fixed-parameter algorithms for kemeny scores. *In Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management, LNCS*, volume 5034, pages 60–71, 2008.

[20] N. Betzler, M. R. Fellows, J. Guo, R. Niedermeier et F. A. Rosamond : How similarity helps to efficiently compute kemeny rankings. *In Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems*, pages 657–664, 2009.

[21] N. Betzler, M.R. Fellows, J. Guo, R. Niedermeier et F.A. Rosamond : Fixed-parameter algorithms for kemeny rankings. *Theoretical Computer Science*, 410(45):4554–4570, 2009.

[22] N. Betzler, J. Guo, C. Komusiewicz et R. Niedermeier : Average parameterization and partial kernelization for computing medians. *Journal of Computer and System Sciences*, 77(4): 774–789, 2011.

[23] T. Biedl, F.J. Brandenburg et X. Deng : Crossings and permutations. *In proceedings of the 13th International Symposium on Graph Drawing (GD 2005), LNCS 3843*, pages 1–12, 2005.

[24] M. Bitan, Y. Gal, S. Kraus, E. Dokow et A. Azaria : Social rankings in human-computer committees. *Proceedings of the 37th AAAI Conference on Artificial Intelligence*, pages 116–122, 2013.

[25] G. Blin, M. Crochemore, S. Hamel et S. Vialette : Finding the median of three permutations under the kendall-tau distance. *In Proceedings of the 7th Permutation Pattern conference*, pages 31–36, 2009.

[26] G. Blin, M. Crochemore, S. Hamel et S. Vialette : Median of an odd number of permutations. *Pure Mathematics and Applications*, 21(2):161–175, 2010.

[27] Borda : Mémoire sur les élections au scrutin. *Histoire de l'Académie Royale des Sciences*, 1781.

[28] B. Brancotte : *Agrégation de classements avec égalités : algorithmes, guides à l'utilisateur et applications aux données biologiques*. Thèse de doctorat, 2015.

[29] B. Brancotte et R. Milosz : Rank aggregation with ties is at least as difficult as rank aggregation without ties. *Internal report, Univ. Paris-Sud - France*, 2015.

[30] B. Brancotte, B. Rance, A. Denise et S. Cohen-Boulakia : Conqur-bio : Consensus rankings with query reformulation for biological data. *In proceedings of the 10th International conference on Data Integration in the Life Sciences*, LNCS 8574:128–142, 2014.

[31] B. Brancotte, B. Yang, G. Blin, A. Denise, S. Cohen-Boulakia et S. Hamel : Rank aggregation with ties : Experiments and analysis. *In Proceedings of the VLDB Endowment (PVLDB)*, volume 8, pages 1202–1213, 2015.

[32] S. Brin et L. Page : The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, 1998.

[33] J. Ceberio, A. Mendiburu et J.A. Lozano : Introducing the mallows model on estimation of distribution algorithms. *In International Conference on Neural Information Processing*, pages 461–470. Springer, 2011.

[34] J. Ceberio, A. Mendiburu et J.A. Lozano : The linear ordering problem revisited. *European Journal of Operational Research*, 241(3):686 – 696, 2015.

[35] S. Chanas et P. Kobylanski : A new heuristic algorithm solving the linear ordering problem. *Computational Optimization and Applications*, 6(2):191–205, 1996.

[36] I. Charon et O. Hudry : A branch-and-bound algorithm to solve the linear ordering problem for weighted tournaments. *Discrete Applied Mathematics*, 154(15):2097 – 2116, 2006.

[37] J. Chen, E.E. Bardes, B.J. Aronow et A.G. Jegga : Toppgene suite for gene list enrichment analysis and candidate gene prioritization. *Nucleic acids research*, 37(suppl_2):W305–W311, 2009.

[38] H.B. Chenery et T. Watanabe : International comparisons of the structure of production. *Econometrica*, 26(4):487–521, 1958.

[39] W. Cohen, R. Schapire et Y. Singer : Learning to order things. *Journal of Artificial Intelligence Research*, 10:243–270, 1999.

[40] S. Cohen-Boulakia, A. Denise et S. Hamel : Using medians to generate consensus rankings for biological data. *In proceedings of th 23rd International Conference on Scientific and statistical Database Management, LNCS 6809*, pages 73–90, 2011.

[41] T. Coleman et A. Wirth : Ranking tournaments : Local search and a new algorithm. *Journal of Experimental Algorithmics*, 14, article 6, 9 pages, 2009.

[42] J.M. Colomer : Ramon llull : From "ars electionis" to social choice theory. *Social Choice and Welfare*, 40(2):317–328, 01 2013.

[43] Condorcet : Essai sur l'application de l'analyse à la probabilité des décisions rendue à la pluralité des voix. *Paris : Imprimerie royale*, 1785.

[44] V. Conitzer, A. Davenport et J. Kalagnanam : Improved bounds for computing kemeny rankings. *In Proceedings of the 21st National Conference on Artificial Intelligence*, pages 620–626, 2006.

[45] A. H. Copeland : A reasonable social welfare function. *University of Michigan Seminar on Applications of Mathematics to the social sciences*, 1951.

[46] B. Correal et P. Galinier : On the complexity of searching the linear ordering problem neighborhoods. *In proceedings of the 15th European Conference on Evolutionary Computation in Combinatorial Optimization*, 9026(1):150–159, 2015.

[47] D. E. Critchlow : Metric methods for analyzing partially ranked data. *Lecture Notes in Statistics*, 34, 216 pages, 1980.

[48] A. D'Ambrosio, S.A. Amodio et C. Iorio : Two algorithms for finding optimal solutions of the kemeny rank aggregation problem for full rankings. *Electronic Journal of Applied Statistical Analysis*, 8(2):198–213, 2015.

[49] A. D'Ambrosio, G. Mazzeo, C. Iorio et R. Siciliano : A differential evolution algorithm for finding the median ranking under the kemeny axiomatic approach. *Computers and Operations Research*, 82:126 – 138, 2017.

[50] A. Dasdan, C. Drome et S. Kolay : Thumbs-up : A game for playing to rank search results. *In Proceedings of the 18th International Conference on World Wide Web*, pages 1071–1072, 2009.

[51] A. Davenport et J. Kalagnanam : A computational study of the kemeny rule for preference aggregation. *In Proceedings of the 19th National Conference on Artificial Intelligence*, pages 697–702, 2004.

[52] R. P. DeConde, S. Hawley, S. Falcon, N. Clegg, B. Knudsen et R. Etzioni : Combining results of microarray experiments : a rank aggregation approach. *Statistical Applications in Genetics and Molecular Biology*, 5(1), Article 15, 2006.

[53] C. Desharnais et S. Hamel : Automedians sets of permutation : extended abstract. *In Proceedings of PP2017 (15th International Conference on Permutation Patterns 2017)*, 5 pages, 2017.

[54] P. Diaconis et R.L. Graham : Spearman's footrule as a measure of disarray. *Journal of the Royal Statistical Society, Series B*, 39(2):262–268, 1977.

[55] R.G. Downey et M.R. Fellows : *Parameterized complexity*. Springer Publishing Company, 533 pages, 2012.

[56] C. Dwork, R. Kumar, M. Naor et D. Sivakumar : Rank aggregation methods for the web. *In proceedings of the 10th International Conference on World Wide Web*, pages 613–622, 2001.

[57] C. Dwork, R. Kumar, M. Naor et D. Sivakumar : Rank aggregation revisited. rapport interne, 19 pages, 2001.

[58] E.J. Emond et D.W. Mason : A new rank correlation coefficient with application to the consensus ranking problem. *Journal of Multi-Criteria Decision Analysis*, 11(1):17–28, 2002.

[59] R. Fagin, R. Kumar, M. Mahdian, D. Sivakumar et E. Vee : Comparing and aggregating rankings with ties. *In Proc. of the 23rd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 47–58, 2004.

[60] R. Fagin, R. Kumar, M. Mahdian, D. Sivakumar et E. Vee : Comparing partial rankings. *SIAM Journal on Discrete Mathematics*, 20(3):628–648, 2006.

[61] R. Fagin, R. Kumar et I.D. Sivakumar : Efficient similarity search and classification via rank aggregation. *In Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 301–312, 2003.

[62] D.S. Felsenthal et H. Nurmi : Two types of participation failure under nine voting methods in variable electorates. *Public Choice*, 168(1-2):115–135, 2016.

[63] H. Fernau, F.V. Fomin, D. Lokshtanov, M. Mnich, G. Philip et S. Saurabh : Ranking and drawing in subexponential time. *In proceedings of the 21st International Workshop on Combinatorial Algorithms*, pages 337–348, 2011.

[64] R.A. Fisher et F. Yates : Statistical tables for biological, agricultural and medical research, london : Oliver and boyd. 3:26–27, 1948.

[65] M.A. Fligner et J.S. Verducci : Distance based ranking models. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 359–369, 1986.

[66] C.G. Garcia, D. Pérez-Brito, V. Campos et R. Martí : Variable neighborhood search for the linear ordering problem. *Computers and Operations Research*, 33(12):3549 – 3565, 2006.

[67] A. Gibbard : Manipulation of voting schemes : A general result. *Econometrica*, 41(4):587–601, 1973.

[68] F.M. Grieve, A.A.O. Plumb et S.H. Khan : Radiology reporting : a general practitioner's perspective. *The British Journal of Radiology*, 83(985):17–22, 2010.

[69] M. Grötschel, M. Juenger et G. Reinelt : A cutting plane algorithm for the linear ordering problem. *Operations Research*, 32(6):1195–1220, 1984.

[70] A. Guénoche, B. Vandeputte-Riboud et J.B. Denis : Selecting varieties using a series of trials and a combinatorial ordering method. *Agronomie*, 14(6):363–375, 1994.

[71] S. Hamel et R. Milosz : Medians of permutations : when A = M(A). *In Procceedings of 12th International Permutation Patterns Conference*, pages 68–71, 2014.

[72] E. Hemaspaandra, H. Spakowski et J. Vogel : The complexity of kemeny elections. *Theoretical Computer Science*, 349(3):382–391, 2005.

[73] B. N. Jackson, P. S. Schnable, et S. Aluru : Consensus genetic maps as median orders from inconsistent sources. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 5(2):161–171, 2008.

[74] R.M. Karp : Reducibility among combinatorial problems. *Complexity of Computer Communications*, pages 85–103, 1972.

[75] M. Karpinski et W. Schudy : Faster algorithms for feedback arc set tournament, kemeny rank aggregation and betweenness tournament. *In proceeding of the 21st International Symposium on Algorithms and Computation, LNCS 6506*, pages 3–14, 2010.

[76] J.G. Kemeny : Mathematics without numbers. *Daedalus*, 88(4):577–591, 1959.

[77] J.G. Kemeny et J.L. Snell : Mathematical models in the social sciences. *Blaisdell Publishing Company*, 145 pages, 1962.

[78] M.G. Kendall : A new measure of rank correlation. *Biometrika*, 30(1-2):81–93, 1938.

[79] M.G. Kendall : The treatment of ties in ranking problems. *Biometrika*, 33(3):239–251, 1945.

[80] C. Kenyon-Mathieu et W. Schudy : How to rank with few errors. *In proceeding of the 38th annual ACM Symposium on Theory of Computing*, pages 95–103, 2007.

[81] M. Kim, F. Farnoud et O. Milenkovic : Hydra : gene prioritization via hybrid distance-score rank aggregation. *Bioinformatics*, 31(7):1034–1043, 2015.

[82] D.E. Knuth : Seminumerical algorithms. *The Art of Computer Programming, Volume 2*, pages 124–125, 1969.

[83] H.W. Kuhn : The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.

[84] R.E. Ladner, N.A. Lynch et A.L. Selman : A comparison of polynomial time reducibilities. *Theoretical Computer Science*, 1(2):103 – 123, 1975.

[85] M. Laguna, R. Marti et V. Campos : Intensification and diversification with elite tabu search solutions for the linear ordering problem. *Computers and Operations Research*, 26(12):1217 – 1230, 1999.

[86] R. Llull : Ars electionis. 1299.

[87] R.D. Luce : *Individual Choice and Behavior : A Theoretical Analysis*. New York : Dover Publications, 176 pages, 2005.

[88] D. Maglott, J. Ostell, K.D. Pruitt et T.Tatusova : Entrez gene : gene-centered information at ncbi. *Nucleic acids research*, 39(sp1):D52–D57, 2011.

[89] C.L. Mallows : Non-null ranking models. i. *Biometrika*, 44(1-2):114–130, 1957.

[90] R. Martí et G. Reinelt : *The Linear Ordering Problem : exact and heuristic methods in combinatorial optimization*. Springer, Applied Mathematical Sciences 175, 172 pages, 2011.

[91] N. Mattei et T. Walsh : Preflib : A library of preference data. *Proceedings of 3rd International Conference on Algorithmic Decision Theory (ADT 2013), LNCS 8176*, pages 259–270, 2013.

[92] M. Meila, K. Phadnis, A. Patterson et J. Bilmes : Consensus ranking under the exponential model. *In Proceedings of the 23rd Annual Conference on Uncertainty in Artificial Intelligence*, pages 285–294, 2007.

[93] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, N. Marshall, A.H. Teller et E. Teller : Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[94] B. Miller, P. Hemmer, M. Steyvers et M.D. Lee : The wisdom of crowds in ordering problems. *In Proceedings of the ninth international conference on cognitive modeling*, 6 pages, 2009.

[95] R. Milosz : Étude combinatoire et algorithmique de médianes de permutations sous la distance de kendall-tau. *Rapport interne, Universite de Montreal*, 81 pages, 2017.

[96] R. Milosz et S. Hamel : Medians of permutations : Building constraints. *In proceedings of the 2nd Conference on Algorithms and Discrete Applied Mathematics, LNCS 9602*, pages 264–276, 2016.

[97] R. Milosz et S. Hamel : Heuristic, branch-and-bound solver and improved space reduction for the median of permutations problem. *In Proceedings of the 28th International Workshop On Combinatorial Algorithms, LNCS 10765*, pages 299–311, 2018.

[98] R. Milosz et S. Hamel : Space reduction constraints for the median of permutations problem. *accepté à Journal of Discrete Applied Mathematics*, 23 pages, mars 2018.

[99] R. Milosz et S. Hamel : Exploring the median of permutations problem. *accepté à Journal of Discrete Algorithms*, 28 pages, octobre 2018.

[100] R. Milosz, S. Hamel et A. Pierrot : Median of 3 permutations, 3-cycles and 3-hitting set problem. *In Proceedings of the 28th International Workshop On Combinatorial Algorithms, LNCS 10979*, pages 224–236, 2018.

[101] K. O'L. Morgan et S. Morgan : *State rankings 2014 : A statistical view of America*. CQ Press, 624 pages, 2014.

[102] G. Munda : Choosing aggregation rules for composite indicators. *Social Indicators Research*, 109(3):337–354, 2012.

[103] X. Munoz : Fair ranking in dancesport competitions. *Proceedings of Permutation Patterns*, pages 39–48, 2003.

[104] N. Nishimura et N. Simjour : Parameterized enumeration of (locally-) optimal aggregations. *In proceedings of the 13th International Symposium on Algorithms and Data Structures, LNCS 8037*, pages 512–523, 2013.

[105] L. Page : Pagerank : Bringing order to the web. *Stanford Digital Library Project, talk*, 1997.

[106] L. Page, S. Brin, R. Motwani et T. Winograd : The pagerank citation ranking : Bringing order to the web. Rapport technique, Stanford InfoLab, 1999.

[107] J. Perrie, Y. Hao, Z. Hayat, R. Colak, K. Lyons, S. Vembu et S. Molyneux : Implementing recommendation algorithms in a large-scale biomedical science knowledge base. *CoRR*, abs/1710.08579, 21 pages, 2017.

[108] R. L. Plackett : The analysis of permutations. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 24(2):193–202, 1975.

[109] A.A. Plumb, F.M. Grieve et S.H. Khan : Survey of hospital clinicians' preferences regarding the format of radiology reports. *Clinical Radiology*, 64(4):386 – 394, 2009.

[110] M. Pujari et R. Kanawati : Supervised rank aggregation approach for link prediction in complex networks. *Proceedings of the 21st International Conference on World Wide Web*, pages 1189–1196, 2012.

[111] D.W. Ritchie et V. Venkatraman : Ultra-fast fft protein docking on graphics processors. *Bioinformatics*, 26(19):2398–2405, 2010.

[112] A. Ruiz-Padillo, T.B.F. de Oliveira, M. Alves, A.L.C. Bazzan et D.P. Ruiz : Social choice functions : A tool for ranking variables involved in action plans against road noise. *Journal of Environmental Management*, 178:1 – 10, 2016.

[113] M.P. Vecchi S. Kirkpatrick, C.D. Gelatt : Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[114] D.G. Saari et V.R. Merlin : A geometric examination of kemeny's rule. *Social Choice and Welfare*, 17(3):403–438, 2000.

[115] M.A. Satterthwaite : Strategy-proofness and arrow's conditions : Existence and correspondence theorems for voting procedures and social welfare functions. *Journal of Economic Theory*, 10(2):187 – 217, 1975.

[116] E.W. Sayers, T. Barrett, D.A. Benson et al. : Database resources of the national center for biotechnology information. *Nucleic acids research*, 40(Database Issue):D13–D25, 2012.

[117] F. Schalekamp et A. van Zuylen : Rank aggregation : Together we're strong. *In Proceedings of the 11th SIAM Workshop on Algorithm Engineering and Experiments*, pages 38–51, 2009.

[118] T. Schiavinotto et T. Stützle : The linear ordering problem : Instances, search space analysis and algorithms. *Journal of Mathematical Modelling and Algorithms*, 3(4):367–402, 2005.

[119] M. Schulze : A new monotonic, clone-independent, reversal symmetric, and condorcet-consistent single-winner election method. *Social Choice and Welfare*, 36(2):267–303, 2011.

[120] D. Sengupta, A. Pyne, U. Maulik et S. Bandyopadhyay : Reformulated kemeny optimal aggregation with application in consensus ranking of microrna targets. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 10(3):742–751, 2013.

[121] J. Sese et S. Morishita : Rank aggregation method for biological databases. *Genome Informatic*, 12:506–507, 2001.

[122] C.E. Shannon : A mathematical theory of communication. *The Bell System Technical Journals*, 27(3):379–423, 1948.

[123] N. Simjour : Improved parameterized algorithms for the kemeny aggregation problem. *In proceedings of the 4th International Workshop on Parameterization and Exact Computation, LNCS 5917*, pages 312–323, 2009.

[124] D. Simon : *Evolutionary Optimization Algorithms*. Wiley, 772 pages, 2013.

[125] J.H. Smith : Aggregation of preferences with variable electorate. *Econometrica*, 41(6):1027–1041, 1973.

[126] S. Soundarajan, T. Eliassi-Rad et B. Gallagher : A guide to selecting a network similarity method. *Proceedings of the 2014 SIAM International Conference on Data Mining*, pages 1037–1045, 2014.

[127] J. Starlinger, B. Brancotte, S. Cohen-Boulakia et U. Leser : Similarity search for scientific workflows. *Proceedings of the VLDB Endowment*, 7(12):1143–1154, 2014.

[128] N. Streib, S.J. Young et J. Sokol : A major league baseball team uses operations research to improve draft preparation. *Interfaces*, 42(2):119–130, 2012.

[129] K. Subbian et P. Melville : Supervised rank aggregation for predicting influencers in twitter. *In 2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing*, pages 661–665, 2011.

[130] T. N. Tideman : Independence of clones as a criterion for voting rules. *Social Choice and Welfare*, 4(3):185–206, 1987.

[131] L.-C. Tranchevent, A. Ardeshirdavani, S. ElShal, D. Alcaide, J. Aerts, D. Auboeuf et Y. Moreau : Candidate gene prioritization with endeavour. *Nucleic acids research*, 44(W1): W117–W121, 2016.

[132] A.R. Travis, M. Sevenster, R. Ganesh, J.F. Peters et P.J. Chang : Preferences for structured reporting of measurement data. *Academic Radiology*, 21(6):785–796, 2014.

[133] M. Truchon : An extension of the condorcet criterion and kemeny orders. *Internal Report, Université Laval*, 16 pages, 1998.

[134] M. Truchon : Aggregation of rankings in figure skating. *CIRPEE Working Paper*, no 04-14, 57 pages, 2004.

[135] A. van Zuylen et D.P. Williamson : Deterministic pivoting algorithms for constrained ranking and clustering problems. *Mathematics of Operations Research*, 34(3):594–620, 2009.

[136] H.S. Wilf : Generatingfunctionology,. *Academic Press, New York*, 147 pages, 1990.

[137] H.P. Young : Social choice scoring functions. *SIAM Journal on Applied Mathematics*, 28(4):824–838, 1975.

[138] H.P. Young : Condorcet's theory of voting. *The American Political Science Review*, 82(4):1231–1244, 1988.

[139] H.P. Young : Optimal voting rules. *Journal of Economic Perspectives*, 9(1):51–64, 1995.

[140] H.P. Young et A. Levenglick : A consistent extension of condorcet's election principle. *SIAM Journal on Applied Mathematics*, 35(2):285–300, 1978.

Annexe A

ANNEXE A : PREUVES ADDITIONNELLES

Définition A.0.1. *La permutation* **renversée** π^r *d'une permutation* $\pi = \pi_1,\pi_2,...,\pi_{n-1},\pi_n$ *est définie telle que* $\pi^r = \pi_n,\pi_{n-1},...,\pi_2,\pi_1$.

Soit $\mathcal{A}$ un ensemble de permutations, alors $\mathcal{A}^r = \{\pi^r | \pi \in \mathcal{A}\}$. On a aussi que $R(\mathcal{A}^r) = R^t(\mathcal{A}) = L(\mathcal{A})$ car le nombre de permutations dans lesquelles i était avant j est maintenant le nombre de permutations ou j est avant i.

Proposition A.0.1. *Soit* $\mathcal{A}$ *un ensemble de* m *permutations de taille* n *et soit* π *une permutation de taille* n*, le score de Kemeny de* π *vers* $\mathcal{A}$ *renversé* (A^r) *est :* $K(\pi, \mathcal{A}^r) = \frac{n(n-1)}{2}m - K(\pi, \mathcal{A})$.

Preuve :

$$K(\pi, \mathcal{A}^r) = \sum_{\substack{i,j\in[n] \\ i\prec_\pi j}} R_{ij}(\mathcal{A}^r)$$

$$= \sum_{\substack{i,j\in[n] \\ i\prec_\pi j}} L_{ij}(\mathcal{A})$$

$$= \sum_{\substack{i,j\in[n] \\ i\prec_\pi j}} m - R_{ij}(\mathcal{A})$$

$$= \frac{n(n-1)}{2}m - \sum_{\substack{i,j\in[n] \\ i\prec_\pi j}} R_{ij}(\mathcal{A})$$

$$= \frac{n(n-1)}{2}m - K(\pi, \mathcal{A}).$$

■

Proposition A.0.2. *Soit* $\mathcal{A}$ *un ensemble de* m *permutations de taille* n *et soit* π *une permutation de taille* n*, le score de Kemeny de* π *renversée* (π^r) *vers* $\mathcal{A}$ *est :* $K(\pi^r, \mathcal{A}) = \frac{n(n-1)}{2}m - K(\pi, \mathcal{A})$.

Preuve :

$$
\begin{aligned}
K(\pi^r, \mathcal{A}) &= \sum_{\substack{i,j\in[n] \\ i\prec_{\pi^r} j}} R_{ij}(\mathcal{A}) \\
&= \sum_{\substack{i,j\in[n] \\ j\prec_{\pi} i}} R_{ij}(\mathcal{A}) \\
&= \sum_{\substack{i,j\in[n] \\ i\prec_{\pi} j}} R_{ji}(\mathcal{A}) \\
&= \sum_{\substack{i,j\in[n] \\ i\prec_{\pi} j}} L_{ij}(\mathcal{A}) \\
&= \sum_{\substack{i,j\in[n] \\ i\prec_{\pi} j}} m - R_{ij}(\mathcal{A}) \\
&= \frac{n(n-1)}{2} m - \sum_{\substack{i,j\in[n] \\ i\prec_{\pi} j}} R_{ij}(\mathcal{A}) \\
&= \frac{n(n-1)}{2} m - K(\pi, \mathcal{A}).
\end{aligned}
$$

■

Corollaire A.0.1. *Soit $\mathcal{A}$ un ensemble de m permutations de taille n et soit π une permutation de taille n alors $K(\pi^r, \mathcal{A}) = K(\pi, \mathcal{A}^r) = \frac{n(n-1)}{2} m - K(\pi, \mathcal{A})$.*

Annexe B

ANNEXE B : ARTICLE POUR JOURNAL OF DISCRETE ALGORITHMS 2018

Exploring the Median of Permutations Problem[☆]

Robin Milosz and Sylvie Hamel

DIRO - Université de Montréal, Canada

Abstract

Given a set $\mathcal{A} \subseteq \mathbb{S}_n$ of m permutations of $\{1, 2, \ldots, n\}$ and a distance function d, the **median** problem consists of finding a permutation π^* that is the "closest" of the m given permutations. Here, we study the problem under the Kendall-τ distance which counts the number of order disagreements between pairs of elements of permutations. In this article, which is an extended version of the conference paper [1], we explore this NP-complete problem using four different approaches: a well parameterized heuristic, an improved search space reduction technique, a refined branch-and-bound solver and an integer linear programming approach using IBM CPLEX. All approaches are tested on synthetic and real life data sets. Combining those approaches, we were able to solve the "ED-00015-00000004.soc" instance from PrefLib which, to our knowledge, was never achieved before.

Keywords: Median of permutations, Kendall-τ distance, simulated annealing, space reduction, branch-and-bound, integer linear programming.

1. Introduction

The problem of finding medians of a set of m permutations of $\{1, 2, \ldots, n\}$ under the Kendall-τ distance [2], often cited as the Kemeny Score Problem [3], consists of finding a permutation that agrees the most with the order of the m given permutations, *i.e.*, that minimizes the sum of order disagreements between pairs of elements of permutations. This problem has been proved to be NP-complete when $m \geq 4$, m even (first proved in [4], then corrected in [5]). Really recently, it as been proved that the problem stays hard for sets of $m \geq 7$ permutations, m odd [6], while the theoretical complexity of the cases $m = 3$ and $m = 5$ remains open. A lot of work as been done in the last 15 years, either on deriving approximation algorithms [7, 8, 9] or fixed-parameter ones [10, 11, 12]. Other theoretical approaches aiming at reducing the search space for this problem have also been developed [13, 14, 15, 16, 17]. Two good comparison works

[☆]This work is supported by a grant from the National Sciences and Engineering Research Council of Canada (NSERC) through an Individual Discovery Grant RGPIN-2016-04576 (Hamel) and by Fonds Nature et Technologies (FRQNT) through a Doctoral scholarship (Milosz).

were done on available algorithms, heuristics and data sets in [18, 19]. We refer the reader to [13] whose authors did an inspiring great work in combining data reduction techniques and exact solvers. Our work is following this perspective.

In this present work, we are interested in solving methods for the median of permutations problem, focusing on four different approaches. Note that this article is an extended version of the conference paper [1]. In this version, the added forth approach, an integer linear programming approach using CPLEX is described in Section 5.2. We also evaluated all our approaches on real data sets from PrefLib [20] (included in added Sections 3.4, 4.6 and 5.3). Furthermore, all of our algorithms are now described by pseudocodes (see Algorithms 1 to 4), new sections are added (Section 3.3 which details the choice of parameters in our simulated annealing heuristic, Section 4.5 in which we analyse the complexity of our search space reduction technique and Section 4.7 in which we combine our method with another search space reduction technique) and the efficiency of our search space reduction approach is tested on more sets of randomized permutations and is presented in the new Tables 10 and 11.

The article is organized as follows: After introducing some basic definitions and notations in Section 2.1 and introducing our data sets in Section 2.2, we present our first approach in Section 3, an adaptation of the well known "Simulated Annealing" heuristic to our context. Second, in Section 4, we built ordering constraints for pairs of elements appearing in a median by merging previous approches [15, 17] complemented by our simulated annealing heuristic, thus reducing significantly the search space for a median. Third, we present, in Section 5, the implementations of two exact solvers: a branch-and-bound algorithm and an integer linear programming approach that are both powered by the two previous approaches. Finally, Section 6 gives some thoughts on future works.[1]

2. Preliminaries

2.1. Definitions and notations

A **permutation** π is a bijection of $[n] = \{1, 2 \ldots, n\}$ onto itself. The set of all permutations of $[n]$ is denoted $\mathbb{S}_n$. As usual we denote a permutation π of $[n]$ as $\pi = \pi_1\pi_2 \ldots \pi_n$, and a segment of π by $\pi[i..j] = \pi_i\pi_{i+1}...\pi_{j-1}\pi_j$. The cardinality of a set $\mathcal{S}$ will be denoted $\#\mathcal{S}$.

The **Kendall-τ distance**, denoted d_{KT}, counts the number of order disagreements between pairs of elements of two permutations and can be defined formally as follows: for permutations π and σ of $[n]$, we have that $d_{KT}(\pi, \sigma) = \#\{(i, j)|i < j \text{ and } [(\pi[i] < \pi[j] \text{ and } \sigma[i] > \sigma[j]) \text{ or } (\pi[i] > \pi[j] \text{ and } \sigma[i] < \sigma[j])]\}$, where $\pi[i]$ denotes the position of integer i in permutation π.

Given any set of permutations $\mathcal{A} \subseteq \mathbb{S}_n$ and a permutation $\pi \in \mathbb{S}_n$, we have $d_{KT}(\pi, \mathcal{A}) = \sum_{\sigma\in\mathcal{A}} d_{KT}(\pi, \sigma)$. The **problem of finding a median of**

[1] The Java code of all our algorithms (7000+ lines) can be found online on GitHub at: https://github.com/robinmilosz/ExploringMedianOfPermutations

$\mathcal{A}$ **under the Kendall-τ distance** can be stated formally as follows: Given $\mathcal{A} \subseteq \mathbb{S}_n$, we want to find a permutation π^* of $\mathbb{S}_n$ such that $d_{KT}(\pi^*, \mathcal{A}) \leq d_{KT}(\pi, \mathcal{A}),\ \forall\ \pi \in \mathbb{S}_n$. Note that a set $\mathcal{A}$ can have more than one median. Note also that, in what follows, the cardinality of $\mathcal{A}$ will always be denoted by m.

To keep track of the number of permutations in $\mathcal{A}$ that have a certain order between two elements, let us introduce the left/right distance matrices L and R.

Definition 1. *Let $L(\mathcal{A})$, be the* **left distance matrix** *of a set of m permutations $\mathcal{A} \subseteq \mathbb{S}_n$, where $L_{ij}(\mathcal{A})$ denotes the number of permutations of $\mathcal{A}$ having element i to the left of element j. Symmetrically, let $R(\mathcal{A})$, be the* **right distance matrix** *of $\mathcal{A}$, where $R_{ij}(\mathcal{A})$ denotes the number of permutations of $\mathcal{A}$ having element i to the right of element j. Obviously, $L_{ij}(\mathcal{A}) + R_{ij}(\mathcal{A}) = m$ and $L_{ij}(\mathcal{A}) = R_{ji}(\mathcal{A})$.*

We can calculate the distance between a permutation $\pi \in \mathbb{S}_n$ and a set of permutations $\mathcal{A} \subseteq \mathbb{S}_n$ using the right (or left) distance matrix as follow:

$$d_{KT}(\pi, \mathcal{A}) = \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i \\ \pi[j] > \pi[i]}}^{n} R_{ij}(\mathcal{A}) = \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i \\ \pi[j] > \pi[i]}}^{n} L_{ji}(\mathcal{A}).$$

2.2. Data sets

In this work we evaluate our four approaches on synthetic and real life data sets. The synthetic data sets are randomly uniformly generated using the Fisher-Yates shuffle [21]. We claim that these instances are among the most difficult to tackle for a few reasons. First, note that the random uniform model is equivalent to the Mallows phi model [22] with dispersion parameter set to 0 [18]. The Mallows phi model is a probabilistic model on permutations, analogous to the gaussian model on numbers, with a central "mode" permutation and with a dispersion parameter $\theta \in \mathbb{R}_+$. When θ is small, the permutations are more dispersed from the central permutation. As θ grows bigger, the permutations become concentrated around a central permutation. Second, it is shown in [18] and [19] that as θ gets smaller (less consensus), the instances become more difficult to solve. Finally, we show in our experiments, that those instances are very difficult to tackle compared to the real life instances.

We took our real life data sets from PrefLib [20] (http://www.preflib.org/). We chose the "soc" data type (strict orders, complete rankings) which are representing permutations multi-sets. There are 315 "soc" files, ranging from sets of cardinality $m = 4$ to $m = 5000$ of permutations of size $n = 3$ to $n = 242$. For better comprehension, we separate those instances in four categories: 1) "easy" for small n ($n < 5$), 2) "medium" for $5 \leq n < 60$, 3) "big" for $60 \leq n < 170$ and 4) "giant" for $n \geq 170$. We calculated the estimation of the dispersion parameter θ from the Mallows phi model for each instance using the method described in [23].

3. A heuristic approach

Our first approach is based on the well known Simulated Annealing (SA) heuristic. This approach will give us an approximative solution for the median problem, an upper bound on the distance we are trying to minimize ($d_{KT}(\pi, \mathcal{A})$) and a direction for our branch-and-bound search.

3.1. Simulated Annealing

Simulated annealing (SA) is a probabilistic meta heuristic for locating a good approximation to the global optimum of a given function in a large search space [24]. It is an adaptation of the Metropolis-Hasting algorithm [25] and works best on large discrete space. For that reason, it is a good choice in our case where the search space is $\mathbb{S}_n$, the space of all permutations of $[n]$.

In a simulated annealing heuristic, each point s of the search space corresponds to a state of a physical system, and minimizing a function $f(s)$ corresponds to minimizing the internal energy of the system in state s. The goal is to bring the system, from a randomly chosen initial state, to a state with the minimum possible energy. At each step, the SA heuristic considers a neighbour s' of the current point s and (1) always moves to it if $f(s') < f(s)$ or (2) moves to it with a certain probability if $f(s') \geq f(s)$. This probability to do a "bad" move is almost 1 at the beginning of the heuristic when the system is heated but goes to 0 as the system is cooled. The heuristic stops either when the system reaches a good enough solution or when a certain number of moves has been made.

In our context, we are given a set of permutations $\mathcal{A} \subseteq \mathbb{S}_n$ for which we want to find a median. The function f we need to minimize, given the set $\mathcal{A}$, is $f(\pi) = d_{KT}(\pi, \mathcal{A})$, for $\pi \in \mathbb{S}_n$. We first choose randomly a starting point in our space i.e. a permutation $\pi \in \mathbb{S}_n$. This permutation is uniformly generated using the Fisher-Yates shuffle [21]. To find neighbours of a permutation π in our system, we consider circular moves defined as follows:

Definition 2. *Given $\pi \in \mathbb{S}_n$, we call* **circular move** *of a segment $\pi[i..j]$ of π, denoted $c[i,j](\pi)$, the cycling shifting of one position to the right, if $i < j$, of this segment inside the permutation π: $c[i,j](\pi) = \pi_1 \dots \pi_{i-1}\pi_{\mathbf{j}}\pi_{\mathbf{i}} \dots \pi_{\mathbf{j-1}}\pi_{j+1} \dots \pi_n$ (if $i > j$, we shift to the left: $c[j,i](\pi) = \pi_1 \dots \pi_{j-1}\pi_{\mathbf{j+1}} \dots \pi_{\mathbf{i}}\pi_{\mathbf{j}}\pi_{i+1} \dots \pi_n$).*

So, to choose a neighbour for our current permutation π, our SA heuristic first randomly generates two integers $i \neq j$, $1 \leq i, j \leq n$, and compute neighbour(π)= $c[i,j](\pi)$, if $i < j$ or neighbour(π)=$c[j,i](\pi)$, otherwise. To decide whether or not we move into state neighbour(π), we compute the difference of energy, noted ΔE, which in our case is the difference $d_{KT}(\text{neighbour}(\pi)), \mathcal{A}) - d_{KT}(\pi, \mathcal{A})$. If this difference is negative or null, the state neighbour(π) is closer to the median of $\mathcal{A}$ and we move to this state. If it is positive, we move to state neighbour(π) depending on the acceptance probability $e^{-\Delta E/T}$, where T is the temperature of the system. This process of going to neighbours states is repeated a fixed number of times during which the permutation with the lowest

energy (distance to $\mathcal{A}$) are kept in a set. This set is returned at the end of the process. Algorithm 1, gives the pseudocode of our heuristic SA.

Algorithm 1: Simulated Annealing($\mathcal{A}$, nbRuns, nbMvts, α)

```
Data: a set A of permutations, the number of repetitions of SA, nbRuns, the number of
      movements allowed, nbMvts, and the cooling factor α
Result: The set BestM containing the best medians found i.e. the permutations π
        minimizing d_KT(π, A)
begin
    n ⟵ number of elements in the permutations of A
    m ⟵ number of permutations in A
    BestM ⟵ {}      // set of best medians found
    EMin ⟵ ∞      // minimal energy of the system found so far
    π ⟵ RandomPermutation(n)      // random starting point
    for i ← 1 to nbRuns do
        T ⟵ (0.25m + 4.0)n      // beginning temperature of the system
        E ⟵ d_KT(π, A)      //current energy of the system
        EMin ⟵ E
        for j ← 1 to nbMvts do
            k ⟵ random[1, n]
            ℓ ⟵ random[1, n]      // repeat till ℓ ≠ k
            neighbour ⟵ c[k, ℓ](π)
            ΔE ⟵ d_KT(neighbour, A) - E      // difference of energy
            if ΔE ≤ 0 then
                π ⟵ neighbour      // good move, move to neighbour
                E ⟵ E + ΔE      // change current energy of system
            else
                rand ⟵ random[0, 1]
                if rand < e^{−ΔE/T} then
                    π ⟵ neighbour      // bad move, accept conditionally
                    E ⟵ E + ΔE      // change current energy of system
                end if
            end if
            if E < EMin then
                BestM ⟵ {π}   // initialize set BestMwith the best consensus
                 found so far
                EMin ⟵ E
            else if E = EMin then
                BestM ⟵ BestM ∪ {π}   // add π to BestM
            end if
            T ⟵ α ∗ T   // cooling the system
        end for
    end for
    return BestM
end
```

3.2. Choosing the parameters

The important parameters of a SA heuristic are: the initial temperature of the system, the cooling factor, the maximal number of movements, the function that propose a neighbouring alternative and the number of repetitions of SA. First, we choose the circular move (see Definition 2) as our neighbouring choosing function. This choice was made for three reasons. First reason, while looking at sets of medians for different sets of permutations $\mathcal{A} \subseteq \mathbb{S}_n$, we observe that most of the medians in a set could be obtained from one another by applying those circular moves. Second reason, we did try a lot of other moves (exchanging element $\pi[i]$ with one of its neighbour $\pi[i-1]$ or $\pi[i+1]$ or exchanging it with any other element $\pi[j]$, inverting the order of the elements of a block, etc.)

that clearly did not converge as quickly as the circular moves. Third reason, the circular move is used in local search heuristic that [18, 19] consider as one of the best heuristics.

For the rest of the SA parameters, note that for each instance of our problem, i.e. a set of permutations $\mathcal{A} \subseteq \mathbb{S}_n$, we have two important parameters: the number m of permutations in $\mathcal{A}$ and the size n of the permutations. This pair (m, n) is very relevant to describe the problem's difficulty. So, we were interested in tuning the SA parameters as function of n and m.

Subsection 3.3 describe the extensive testing that we did to find out the optimal parameters for SA given the pair (m, n). But here is an overview:

1. Initial solution: a random permutation generated by Fisher-Yates shuffle.
2. The cooling schedule is: $t_i = \alpha t_{i-1}$.
3. The initial temperature of the system set to: $t_0 \longleftarrow (0.25m + 4.0)n$
4. The cooling factor set to:

$$\alpha = \begin{cases} 0.99 & \text{if } m = 3, 4 \\ 0.95 & \text{if } n \leq 10 \text{ and } m \geq 5 \\ 0.99 & \text{if } 11 \leq n \leq 16 \text{ and } m \geq 5 \\ 0.999 & \text{if } 17 \leq n \leq 20 \text{ and } m \geq 5 \\ 0.9995 & \text{if } 21 \leq n \leq 24 \text{ and } m \geq 5 \\ 0.9998 & \text{otherwise} \end{cases}$$

5. The neighbour generating function: the circular move.
6. The number of allowed movements for a solution:

$$\text{nbMvts} = \begin{cases} 0.6n^3 - 11n^2 + 127n & \text{if } m = 3 \\ 0.9n^3 - 29n^2 + 435n - 1600 & \text{if } m = 4 \\ 250 & \text{if } n \leq 7 \text{ and } m \geq 5 \\ 80n^2 - 400n - 1000 & \text{if } n \geq 8 \text{ and } m \geq 5 \end{cases}$$

7. The number of times to repeat the SA heuristic:

$$\text{nbRuns} = \begin{cases} \lceil 0.05n + 2 \rceil & \text{if } m = 3, 4 \\ \lceil 0.007nm + 3 \rceil & \text{if } m \text{ is odd} \\ \lceil 0.002mn + 3 \rceil & \text{if } m \text{ is even} \end{cases}$$

The initial temperature was set to accept (almost) all neighbours in the first iterations. The cooling constant and the number of movements were chosen to have a good compromise between the success probability (probability of achieving the optimal score) and the computing time for each (m, n). The number of SA runs was set such that the probability of overall success should be 99.9% in average for all instances $n \leq 38$ and $m \leq 50$. In practice, it is closer to 99% as some instances are very difficult to solve. For greater m and n, we observed a similar SA behaviour with a general tendency to fail at finding the optimum

more often. For exemple, at $n = 60$, SA will be successful in only the 3/4 of executions for some m (see Section 3.4).

As $m = 3, 4$ are unique cases, that do not seem to be affected by the cooling constant in our considered range and seem much easier to optimize, we treated them appart. For the rest, we found out that m odd problems are harder to solve than m even problems, and required more runs to reach the same success rate. Without surprise, as n and m are getting bigger, the problem is harder to solve, n having even a stronger effect than m.

3.3. Extensive testing for choosing the simulated annealing parameters

For the parameters that were not discuss in Section 3.2, here is more information on why and how they were chosen.

Initial solution: We chose to generate a random uniform permutation as the initial solution. This choice permits to start exploring the search space from any point. We tested different initial solution methods like starting from a approximation given by a simple heuristic like Borda [26] or Copeland [27], but no improvement was noticed. We assume that the high initial temperature that we gave to the system, which allows almost any movement in the first steps, may be responsable for that observation as the initial solution is quickly shuffled by many circular moves. Note also that in our implementation, the best permutation found in one SA run is given as input for the next run. Therefore, it can be considered as a very good starting point, even if the high initial temperature can cancel this advantage.

Cooling scheme: Note that in this work, we choose the exponential cooling schedule ($t_n = \alpha t_{n-1}$), where α is the cooling constant and t_n represents the temperature of the system at iteration n. It is the classic cooling scheme proposed in [24] and works well with our problem in practice as our experimental results are showing. Other cooling schedules include linear cooling ($t_n = max\{t_0 - \alpha(n), t_{\min}\}$), inverse cooling ($t_n = t_{n-1}/(1 + \beta t_{n-1})$), logarithmic cooling ($t_n = c/ln(n + d)$) and inverse linear cooling ($t_n = \alpha t_0/n$). We also chose to keep the same cooling schedule for all runs. We tested lower initial temperatures for subsequent runs but no improvement was noticed and it seemed that this was taking away the advantage to explore further the search space, performing sometimes worse on difficult instances.

Initial temperature: The initial temperature of the system was chosen in a way that the worst possible circular movement would be accepted with probability $p = e^{-1} \sim 37\%$ by setting $t_0 = \Delta_{max}$. We ran simulations on 100 sets $\mathcal{A}$ of m permutations of $[n]$ to find, for each pair (m, n), this maximum observed Δ, with 10 SA runs of 10000 movements per instance (see Table 1). The formula $((0.25 \times m + 4.0) \times n)$ approximates the Δ_{max} for each (m, n). The difference between the formula and the observations is ranging from -22% to +75% (see Table 2).

As the maximum rarely occurs (probably in conditions having the current solution close to the median and a neighbour produced by an extreme circular

m/n	10	15	20	25	30	35	40	45	50
3	27	42	57	70	85	96	117	130	139
10	68	88	122	150	214	212	248	294	294
15	75	138	161	170	259	280	365	412	497
20	90	130	182	242	332	312	392	436	476
40	150	198	258	330	386	482	464	644	710
60	156	272	300	416	486	552	642	786	824
80	170	274	340	576	508	670	766	888	1058

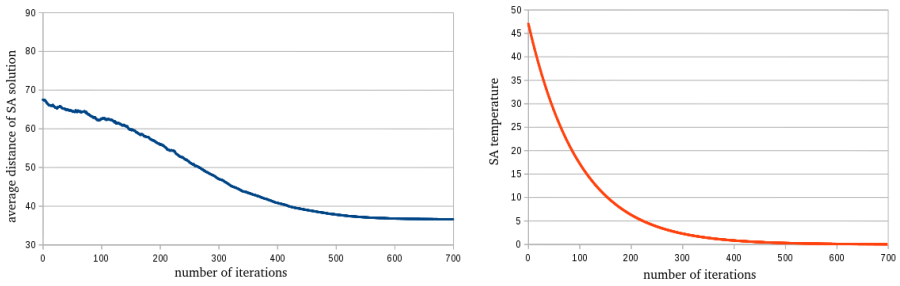
Table 1: Δ_{max} for each pair (m, n). Simulations on 100 sets of m permutations of $[n]$, with 10 SA runs of 10000 movements per instance. Δ_{max} is the biggest observed difference of score between a current solution and its neighbour for each (m, n).

m/n	10	15	20	25	30	35	40	45	50
3	76%	70%	67%	70%	68%	73%	62%	64%	71%
10	-4%	11%	7%	8%	-9%	7%	5%	-1%	11%
15	3%	-16%	-4%	14%	-10%	-3%	-15%	-15%	-22%
20	0%	4%	-1%	-7%	-19%	1%	-8%	-7%	-5%
40	-7%	6%	9%	6%	9%	2%	21%	-2%	-1%
60	22%	5%	27%	14%	17%	20%	18%	9%	15%
80	41%	31%	41%	4%	42%	25%	25%	22%	13%

Table 2: Difference in % between the experimental Δ_{max} of Table 1 and the approximation formula $((0.25\times m+4.0)\times n)$ for each (m, n) i.e. $[\Delta_{max}(m, n)-(0.25\times m+4.0)\times n]/[\Delta_{max}(m, n)]$

movement), we may say that the average Δ have a great chance of being accepted in the first iterations using this approximative formula as initial temperature. Having an initial temperature set too high is unnecessary as the solution will not converge immediately, making the computations useless. Therefore, we are going to set the initial temperature to $t_0 = (0.25 \times m + 4.0) \times n$. You can see in Figures 1a and 1b that experimentations did validate this choice. Figures 1a presents the average distance of a SA solution per iteration for sets $\mathcal{A}$ of $m = 3$ permutations of $[n]$, $n = 10$. For each instance, we ran the SA heuristic and noted at each iteration (from 1 to 700) the distance of the current solution to set $\mathcal{A}$. We then made an average of the distance at each iteration. The corresponding temperature at each iteration can be visualized on Figure 1b. The tendency of the average distance to drop is clearly visible. The most pronounced drop is located approximately between the 150th and the 350th iterations of SA, where solutions are converging at a fast rate towards the optimum. In the first iterations, SA is rather in a exploration phase and after 350 iterations, SA is in the final cooling stage where it refines its current solution.

Optimizing the other parameters: Having set the initial temperature, the cooling schedule and the neighbour function, we made extensive testing (more than 407 hours on 14 Intel i7 @ 2.93GHz processors) to find out the optimal parameters for SA given the pair (m, n). For each triplet (α, m, n), with the cooling factor $\alpha \in \{0.9, 0.95, 0.99, 0.995, 0.998,\ 0.999, 0.9995, 0.9998\}$, the



(a) (b)

Figure 1: (a) The average distance of a SA solution per iteration, where the x-axis represent the number of iterations and the y-axis the average distance. (b) SA temperature, using the exponential cooling scheme with initial temperature set to $(m \times 0.25 + 4) \times n = 47.5$ and $\alpha = 0.99$ cooling constant. Statistics generated on 1000 different sets $\mathcal{A}$ of $m = 3$ permutations of $[n]$, $n = 10$.

number of permutations $m \in \{3, 4, 5, 6, 7, 8, 9, 10,\ 15, 20, 25, 30, 35, 40, 45, 50\}$ and the number of elements $n \in \{8, 10, 12, ..., 36,\ 38\}$, we generated 500 sets of $\mathcal{A}$, solved them using our branch-and-bound to obtain their optimal score, ran the SA heuristic 1000 times on each set, allowing 100000 movements per run. For every SA run, we noted if the heuristic was successful in finding an optimal solution and if so, the number of movements required to reach that solution. Compiling those observations, we noted for each (α, m, n) the average success probability p and $t95$ a threshold on movement such that 95% of the successful runs find the optimum within this number of movements.

A general tendency was that slower cooling (bigger α) gives a better probability of success but requires more movements. Another observable fact is that as m and n gets bigger, the probability of success gets lower, with odd m being even more difficult. Figure 2 and Tables 3 to 6 show the results of our experimentation.

α/n	10	14	16	20	24	28	32	38
0.9	100.00%	100.00%	99.93%	99.72%	98.95%	98.40%	97.24%	94.93%
0.95	100.00%	99.89%	99.75%	99.32%	99.54%	98.00%	97.29%	95.11%
0.99	99.99%	99.92%	99.90%	99.58%	99.13%	98.90%	96.64%	94.07%
0.995	100.00%	99.94%	99.82%	99.77%	99.08%	98.05%	97.55%	94.40%
0.998	100.00%	99.99%	99.92%	99.75%	99.23%	98.44%	96.98%	94.83%
0.999	100.00%	99.95%	99.89%	99.72%	99.47%	98.28%	97.78%	95.00%
0.9995	100.00%	100.00%	99.92%	99.76%	99.62%	98.99%	97.88%	95.13%
0.9998	100.00%	100.00%	100.00%	99.94%	99.63%	99.11%	98.41%	97.08%

Table 3: Average probability of success of SA for each α, $m = 3$ and different n with the number of movements of SA limited to 100000. Based on 500 randomly generated sets $\mathcal{A}$ for each $(\alpha, m = 3, n)$ and 1000 SA runs per set. We can see that the cooling constant has negligible impact for sets of three permutations in this range of n.

Cooling factor: Our choice of the cooling parameter for each (m, n) was first based on the compromise between the number of movements (the time)

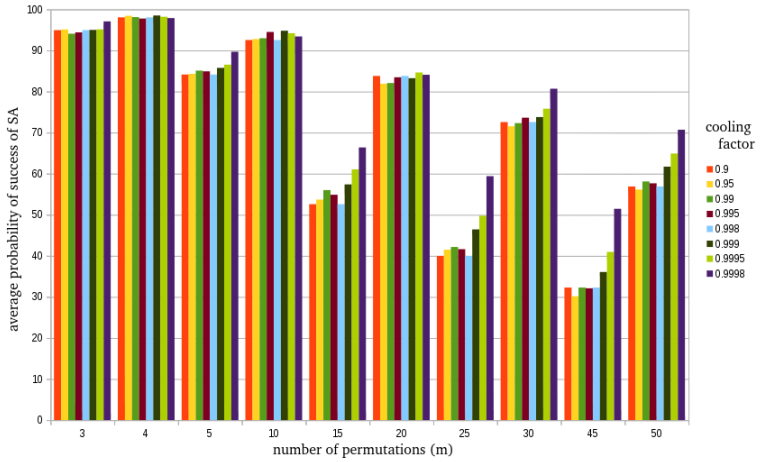


Figure 2: Average probability of success of SA for each α, $n = 38$ and different m with the number of movements of SA limited to 100000. Based on 500 randomly generated sets $\mathcal{A}$ for each m, and 1000 SA runs per set. We can observe that the cooling constant has limited impact on the probability of success for $m = 3, 4$, but greatly affects bigger m.

α/n	10	14	16	20	24	28	32	38
0.9	93.80%	88.70%	82.38%	74.45%	63.07%	54.67%	45.75%	32.28%
0.95	95.22%	87.78%	83.38%	72.77%	63.56%	52.42%	44.05%	30.15%
0.99	96.51%	89.00%	85.71%	74.49%	64.93%	53.69%	45.11%	32.27%
0.995	97.12%	89.48%	85.83%	74.73%	63.56%	53.34%	44.52%	32.10%
0.998	99.13%	93.18%	88.88%	78.42%	69.22%	58.60%	46.71%	33.98%
0.999	99.86%	96.48%	91.73%	83.93%	72.17%	58.90%	49.27%	36.07%
0.9995	99.97%	97.96%	95.88%	89.10%	79.26%	65.23%	55.51%	40.95%
0.9998	100.00%	99.74%	99.03%	94.52%	84.99%	75.49%	66.36%	51.44%

Table 4: Average probability of success of SA for each α, $m = 45$ and different n with the number of movements of SA limited to 100000. Based on 500 randomly generated sets $\mathcal{A}$ for each $(\alpha, m = 45, n)$ and 1000 SA runs per set. The cooling constant has a strong impact for $m = 45$ permutations in bigger values of n.

α/n	10	14	16	20	24	28	32	38
0.9	324	780	1117	2372	4210	6624	11140	19898
0.95	386	813	1192	2431	4005	7538	10369	18821
0.99	709	1170	1533	2648	4311	6735	10864	18845
0.995	1149	1645	1999	2955	4710	7186	11276	19278
0.998	2452	3132	3612	4798	6382	9325	12311	20810
0.999	4543	5670	6192	7277	9169	11880	15472	25337
0.9995	8460	10533	11310	12888	14823	17278	21767	28394
0.9998	19466	24705	26594	29294	31994	34933	38838	44646

Table 5: $t95$ (95% threshold) for sets of three permutations $m = 3$: 95% of the successful SA runs required less than $t95$ movements to converge to a optimal solution. Based on 500 randomly generated sets $\mathcal{A}$ for each $(\alpha, m = 3, n)$ and 1000 SA runs per set. We can observe that for slower cooling (bigger α), SA needs a lot of time to cool down and to achieve an optimal solution.

α/n	10	14	16	20	24	28	32	38
0.9	422	1092	1561	2974	5131	8148	11515	19305
0.95	451	1092	1627	3151	4938	7769	11055	18333
0.99	788	1446	1883	3279	5276	8145	11890	18153
0.995	1267	1895	2310	3721	5936	8189	11870	18291
0.998	2637	3458	3928	5470	7185	9564	13389	20775
0.999	4811	6173	6795	8103	9928	12589	16214	22331
0.9995	8912	11464	12396	14144	15816	18499	21471	28060
0.9998	20199	26429	28630	31953	34954	37311	40086	46491

Table 6: $t95$ (95% threshold) for sets of 45 permutations $m = 45$: 95% of the successful SA runs required less than $t95$ movements to converge to a optimal solution. Based on 500 randomly generated sets $\mathcal{A}$ for each $(\alpha, m = 45, n)$ and 1000 SA runs per set. As in Table 5 we still observe that for slower cooling (bigger α), SA needs a lot of time to cool down and achieve an optimal solution. The difference of $t95$ between $m = 3$ and $m = 45$ is small, meaning that the number of movements required to reach an optimal solution is similar for all m.

and the probability of success and second, on the simplicity and generalizability of the formula. As $m = 3, 4$ are unique cases, that do not seem to be affected by the cooling constant in our considered range (see Figure 2), we treated them appart. Table 7 and Table 8 are respectively giving the probability of success and the $t95$ threshold for each (m, n) and its associated chosen (optimal) α.

m/n	10	16	20	24	28	32	38
3	99.99%	99.90%	99.58%	99.13%	98.90%	96.64%	94.07%
4	100.00%	100.00%	100.00%	99.94%	99.85%	99.57%	98.12%
5	99.85%	98.89%	98.49%	97.22%	96.34%	93.98%	89.68%
10	100.00%	99.83%	99.82%	99.47%	98.68%	97.91%	93.40%
15	97.71%	92.77%	90.88%	88.36%	84.93%	78.38%	66.38%
20	99.95%	99.17%	98.44%	96.77%	96.14%	93.33%	84.07%
25	96.31%	88.21%	87.34%	83.26%	80.05%	70.95%	59.39%
30	99.44%	97.44%	96.92%	94.15%	91.97%	88.34%	80.69%
35	94.97%	86.42%	84.67%	80.53%	77.71%	67.52%	52.50%
40	99.17%	96.44%	95.11%	89.54%	90.36%	83.58%	74.72%
45	95.22%	85.71%	83.93%	79.26%	75.49%	66.36%	51.44%
50	98.84%	95.07%	92.67%	89.97%	86.96%	80.54%	70.69%

Table 7: Average probability of success for SA, with chosen optimal α for each (m, n) and the number of movements of SA limited to 100000. Based on 500 randomly generated sets $\mathcal{A}$ for each (m, n) and 1000 SA runs per set.

Number of movements: Our formulas for nbMvts, the number of movements allowed in our SA heuristic, were chosen such as nbMvts $\geq t95$ the 95% movements threshold for each (m, n) and its corresponding α chosen previously (see Table 8). Simplicity of the formulas was considered but balanced with efficiency considerations. As the choice of the cooling factor was partitioning (m, n), the formulas for the number of movements are also defined on partitions of (m, n).

Number of SA runs: Finally, the number nbRuns, of repetitions of SA was determined so that the average probability of success is around 99.9% for each

m/n	10	16	20	24	28	32	38
3	709	1533	2648	4311	6735	10864	18845
4	574	1253	2360	4177	6791	10421	20672
5	395	1677	7829	15587	36703	40940	51210
10	339	1710	7331	15265	35399	42041	53547
15	403	1936	8089	16120	36605	41512	48986
20	387	1924	8183	16346	37152	43682	55890
25	440	1958	7850	15595	36691	40634	47029
30	432	2059	8531	17305	38905	43381	54617
35	461	1995	7955	15761	37056	40457	46264
40	428	2057	8407	17527	38448	42803	53824
45	451	1883	8103	15816	37311	40086	46491
50	458	2155	8615	16814	38649	45332	53700

Table 8: $t95$ (95% threshold): 95% of the successful SA runs of Table 7 (with optimal chosen α) required less than $t95$ movements to converge to an optimal solution. Based on 500 randomly generated sets $\mathcal{A}$ for each (m, n) and 1000 SA runs per set.

(m, n) in the considered range. In reality, it is a little bit lower because SA runs are not independent and some instances are very difficult to solve, even when using many SA runs. Let p_{t95} be the probability of success with nbMvts movements. The probability of failure is $f = 1 - p_{t95}$. If we run SA nbRuns times, the probability of failure will be f^{nbRuns}. As we want a probability of failure lower than 0.001, we chose a formula for nbRuns such as nbRuns $\geq log(0.001)/log(f) = log(0.001)/log(1 - (p \times 0.95))$ for each (m, n):

$$\text{nbRuns} = \begin{cases} \lceil 0.05n + 2 \rceil & \text{if } m = 3, 4 \\ \lceil 0.007nm + 3 \rceil & \text{if } m \text{ is odd} \\ \lceil 0.002mn + 3 \rceil & \text{if } m \text{ is even.} \end{cases}$$

3.4. Experimental work with Simulated Annealing

We tested our SA heuristic both on the synthetic and real life data sets presented in Section 2.2. As the SA heuristic was parameterized using random uniform data sets with $n \leq 38$ and $m \leq 50$, it gave the same results, as expected, on other random uniform data sets. We tested further on sets of bigger random permutations. For each couple of parameters $n \in \{30, 40, 50, 60, 70\}$ and $m \in \{3, 4, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50\}$, we generated 100 random uniform instances and ran our SA heuristic 10 times on each instance. As it can be observed in Table 9, for $n = 30$, our SA finds the optimal solution (the median) in more than 98% of the cases; for $n = 40$ in more than 94% of the cases, then in more than 89% of the cases for $n = 50$, 76% of the cases for $n = 60$ and 56% of the cases for $n = 70$.

As our SA heuristic was designed using random uniform permutations sets, which are difficult sets to solve, it performed very well on real life data sets. From the 315 "soc" instances of PrefLib [20], SA found the optimal solution for all of them except one: the "ED-00015-00000004.soc" instance. This giant

m/n	30	40	50	60	70
3	99.30%	99.50%	98.20%	95.50%	96.30%
4	98.90%	99.70%	99.40%	99.00%	98.10%
5	98.80%	97.10%	96.60%	96.10%	91.00%
10	100.00%	100.00%	98.20%	98.60%	94.30%
15	99.10%	98.50%	91.90%	84.70%	74.20%
20	100.00%	99.90%	97.00%	97.00%	95.80%
25	99.60%	94.30%	89.10%	76.90%	67.40%
30	99.40%	99.20%	99.50%	95.80%	93.60%
35	99.80%	97.60%	89.00%	77.00%	66.10%
40	99.90%	99.60%	97.20%	93.00%	89.10%
45	99.60%	98.20%	89.90%	77.90%	56.60%
50	100.00%	99.10%	97.70%	94.80%	87.30%

Table 9: Statistics on the efficiency of the SA heuristic in % of times where the heuristic found the optimal solution on random uniform sets of m permutations of size n. For each couple (m, n), 100 instances were generated and SA was run independently 10 times on each instance.

instance is consisting of $m = 5$ permutations of $n = 242$ elements. SA gave an approximation permutation with a generalized Kemeny score of 33498. This is really close to the optimal score of 33497, calculated by our reinforced ILP approach of Section 5.2. Note that running SA on the big instances was much quicker than the exact approaches and yielded the same results, only without the guarantee of optimality. Therefore, our SA is a strong heuristic well suited for real life applications.

Finally we compare our heuristic with other heuristics. Note that when the SA heuristic reaches a system temperature of zero, or very close to zero, it becomes equivalent to the local search algorithms [18, 19] and very similar to the FindMedian heuristic of [14]. Thus, it cannot be worse than those heuristics. We implemented the Copeland method [27] followed by a local search [18]. In our experiments, our SA heuristic always gives a result with a better or equal generalized Kemeny score compared to this method. We finally compared our heuristic with the state-of-the-art FUR and SIgFUR heuristics from [28]. On the giant "ED-00015-00000001.soc" instance from PrefLib, $m = 4$ and $n = 240$, SA found a solution with a score of 14459 (which is optimal) after less than 180 seconds as the FUR and SIgFUR heuristics took respectively 1575 and 586 seconds to find solutions with a score of 14463.

4. Space reduction technique

In [17] and [29] we found theoretical properties of a set of permutations $\mathcal{A} \subseteq \mathbb{S}_n$ (called the **Major Order Theorems**) that can solve the relative order between some pairs of elements in median permutations of $\mathcal{A}$ thus reducing the search space. In this section we will show how to find additional constraints for the problem by merging this previous work on constraints with a lower bound idea of Conitzer *et al.*([15]), giving us an even stronger lower bound, that can

be then combined with the upper bound obtained by the simulated annealing method presented in Section 3. But first, let us quickly recalled our major order theorems in Section 4.1 and Conitzer *et al.* lower bound idea in Section 4.2.

4.1. Some constraints

Given a set of permutations $\mathcal{A} \subseteq \mathbb{S}_n$, the major order theorems, presented in [29], solve the relative order between some pairs of elements in median permutations of $\mathcal{A}$. Thus, these theorems build a set of constraints that can be represented as a boolean matrix C, where $C_{ij} = 1$ if and only if we know that element i will precede element j (denoted $i \prec j$) in median permutations of $\mathcal{A}$. Note that this set of constraints reduces the search space for a median by cutting off all permutations breaking at least one constraint.

Here, we resume the ideas behind these theorems[2] but first, let us formally define the major order between pairs of elements.

Definition 3. *Let $\mathcal{A} \subseteq \mathbb{S}_n$ be a set of permutations. Let $L(\mathcal{A})$ and $R(\mathcal{A})$ be the left and right matrices of $\mathcal{A}$ as defined in Definition 1. Given two elements i and j, $1 \leq i < j \leq n$, we say that the* **major order** *between elements i and j is $i \prec j$ (resp. $j \prec i$) if $L_{ij}(\mathcal{A}) > R_{ij}(\mathcal{A})$ (resp. $R_{ij}(\mathcal{A}) > L_{ij}(\mathcal{A})$), the* **minor order** *is then $j \prec i$ (resp. $i \prec j$). We use δ_{ij} to denote the absolute value of the difference between the major and minor order of two elements i and j.*

The idea of the first major order theorem (**MOT1**) relies on the proximity of two elements in permutations of $\mathcal{A}$: if $i \prec j$ is the major order of elements i and j in permutations of $\mathcal{A}$, we can say that i will also be placed before j in all medians of $\mathcal{A}$ if the numbers of elements (including possible copies) between i and j in those permutations of $\mathcal{A}$ where $j \prec i$ is **less than** δ_{ij}. Intuitively, this multi-set of elements between i and j, that we will called **interference multi-set**, act as interference to the major order $i \prec j$ and so, if its cardinality is small enough, MOT1 gives a relative order for the pair (i, j) in medians of $\mathcal{A}$.

The second major order theorem (**MOT2**) built on the first one by reducing the cardinality of the interference multi-set of a pair of elements i and j by removing from it any element that also appears in between i and j in permutations of $\mathcal{A}$ that follows the major order of this pair.

The idea of the third Major Order Theorem (**MOT3**) is to use previously found constraints (MOT1 and MOT2) to reduce even more the cardinality of the interference multi-set of a pair of elements i and j, by removing from it all elements that cannot be in it. As an example, say that an element k is in the interference multi-set of pair (i, j). This means that k appears in between i and j in at least one permutation of $\mathcal{A}$ where i and j are in their minor order. If we have already found using MOT1 or MOT2 that $C_{ki} = C_{kj} = 1$ or that $C_{ik} = C_{jk} = 1$ then k cannot be in between i and j and we can remove it from

[2] More detailed explanations and some examples for these Major Order Theorems can be found in Section 4 of [29].

the interference multi-set. This process is repeated until no new constraint is found.

Example 1. *The MOT3 is better illustrated with the following exemple: for* $\mathcal{A} = \{78\underline{2}36\underline{1}54, 35\underline{1}786\underline{2}4, 5834\underline{1}\underline{2}76\}$*, the major order for* 1 *and* 2 *is* $1 \prec 2$*,* $\delta_{12} = 1$ *and we have* $\{3, 6\}$ *as the interference multi-set. The* 6 *gets cancelled by the* 6 *between* 1 *and* 2 *in the second permutation (MOT2's way) as the* 3 *is eliminated by contraints* $3 \prec 1$ *and* $3 \prec 2$ *which were found by MOT1. Therefore the interference multi-set is empty and* $1 \prec 2$ *is a valid constraint.*

In [17], we extended those major order theorems by considering the equality case (denoted MOTe) *i.e* the extended MOT theorems gives us the relative order of a pair of elements if the cardinality of the interference multi-set for this pair is **less than or equal to** δ_{ij}. This case is more delicate as the method builds constraints only for a subset of the set of median permutations of $\mathcal{A}$, so care is to be taken to avoid possible contradicting constraints (it is possible that in one median of $\mathcal{A}$, $i \prec j$ and in another $j \prec i$; so using the MOTe theorems we will strictly find those medians of $\mathcal{A}$ satisfying one and only one of those "contradicting" constraint). However, the MOTe theorems have a much better efficiency than the MOT theorems, as you can see in Table 10.

4.2. A new lower bound

Given a set of permutations $\mathcal{A} \subseteq \mathbb{S}_n$, Davenport and Kalagnanam [16] propose a first intuitive lower bound on the median problem of $\mathcal{A}$: the sum, for all pairs of elements i, j, $1 \leq i < j \leq n$, of the number of permutations in $\mathcal{A}$ having the pair (i, j) in its minor order. This bound corresponds to the possibility of ordering all pairs in a median of $\mathcal{A}$ with respect to their major order. If this ordering is possible without conflicting pairs, the bound is equal to the Kendall-τ distance of a median to $\mathcal{A}$. The bound can be easily computed by the following formula:

$$LowerBound_0 = \sum_{\substack{i<j \\ i,j \in [n]}} \min\{R_{ij}, R_{ji}\}$$

Consider the directed weighted graph $G = (V, E)$, where each vertices $v \in V$ is a element of $[n]$ and where E is composed of two edges for each pair of vertices i, j: e_{ij} and e_{ji} with respected weights $w(e_{ij}) = R_{ji}(\mathcal{A})$ and $w(e_{ji}) = R_{ij}(\mathcal{A})$. The median problem can be reformulated as a minimum feedback arc set problem [30] which consist of finding the set of edges $E^* \subset E$ of minimal total weight $w(E^*) = \sum_{e \in E^*} w(e)$ to be removed from G to obtain a directed acyclic graph. Let G' be the graph obtained from G by "cancelling", for each pair of vertices, opposing ordering in permutations of $\mathcal{A}$: $w(e'_{ij}) = w(e_{ij}) - \min\{R_{ij}, R_{ji}\}$, $\forall 1 \leq i < j \leq n$. Obviously $w(E^*) = w(E'^*) + LowerBound_0$.

Let $DC_{G'}$ be a set of disjoint directed cycles of G'. The previous lower bound can be augmented by adding for each cycle $c \in DC_{G'}$ the minimal weight of one

a)

m/n	10	15	20	25	30	35	40	45
3	0.718	0.635	0.579	0.537	0.506	0.481	0.461	0.447
5	0.687	0.595	0.530	0.481	0.444	0.413	0.387	0.361
10	0.606	0.524	0.465	0.420	0.384	0.356	0.333	0.310
15	0.693	0.579	0.500	0.442	0.400	0.366	0.340	0.316
20	0.632	0.540	0.472	0.423	0.383	0.351	0.326	0.306
25	0.699	0.581	0.500	0.442	0.398	0.362	0.333	0.312
30	0.647	0.550	0.479	0.426	0.386	0.352	0.323	0.301
35	0.703	0.584	0.501	0.441	0.396	0.363	0.334	0.309
40	0.656	0.556	0.483	0.428	0.388	0.355	0.328	0.307
45	0.706	0.587	0.502	0.443	0.397	0.360	0.335	0.306
50	0.663	0.562	0.486	0.431	0.388	0.354	0.325	0.301

b)

m/n	10	15	20	25	30	35	40	45
3	0.948	0.878	0.808	0.750	0.702	0.665	0.629	0.604
5	0.896	0.800	0.715	0.647	0.595	0.554	0.517	0.490
10	0.930	0.824	0.709	0.616	0.549	0.497	0.452	0.419
15	0.818	0.704	0.612	0.543	0.490	0.444	0.412	0.383
20	0.876	0.747	0.636	0.553	0.493	0.446	0.407	0.377
25	0.796	0.680	0.587	0.518	0.465	0.425	0.392	0.363
30	0.851	0.720	0.610	0.530	0.472	0.428	0.389	0.362
35	0.787	0.668	0.575	0.506	0.453	0.411	0.378	0.345
40	0.836	0.702	0.596	0.519	0.461	0.415	0.384	0.354
45	0.780	0.660	0.568	0.499	0.448	0.406	0.374	0.350
50	0.824	0.691	0.586	0.512	0.456	0.413	0.379	0.352

c)

n	10	15	20	25	30	35	40	45
nbInstances	100000	80000	50000	20000	10000	4000	2000	1000

Table 10: Average efficiency of *a*) MOT and *b*) MOTe constraints in % of resolution of pairs of elements on sets of uniformly distributed random sets of m permutations, $m = 3$ and $m = 5x$, $1 \leq x \leq 10$ of $[n]$, $n = 5x$, $2 \leq x \leq 9$. The table in *c*) gives the number of sets generated for each n.

of its edges:

$$LowerBound_1 = LowerBound_0 + \sum_{c \in DC_{G'}} \min_{e \in c}\{w(e)\}.$$

In [15], Conitzer, Davenport and Kalagnanam push further the lower bound, proving that the cycles do not need to be disjoint. Let $JC_{G'}$ be any sequence $c_1, c_2, ..., c_l$ of G', that can share common edges. Let $I(c, e)$ be an indicator function of the inclusion of an edge e to a cycle c, *i.e.* $I(c, e) = 1$ if $e \in c$ and 0 otherwise. If $v_i = \min_{e \in c_i}\{w(e) - \sum_{j=1}^{i-1} I(c_j, e)v_j\}$, then we obtain the new following lower bound:

$$LowerBound_2 = LowerBound_0 + \sum_{i=1}^{l} v_i.$$

In practice, we can apply this lower bound method by iteratively searching for a

cycle c in G', containing only non-zero weight edges, finding its minimal weight edge e, then subtracting $w(e)$ to the weight of all edges of c and adding it to the lower bound. This process is then repeating until no such cycle is left in G'.

The process of finding the strongest lower bound, *i.e.* the best sequence and choice of cycles, becomes a problem itself and can be resolved using linear programming. As we are interested here by an efficient pre-processing of the problem, we will use a restrained version of this previous lower bound that can be calculated quickly. Thus, only cycles of length 3 (3-cycles) will be considered.

Our contribution resides in taking advantage of a set of constraints (the one described in Section 4.1) while calculating $LowerBound_2$ as it provides additional information on the structure of the optimal solution. If $C_{ij} = 1$ then we know that the order of elements i and j in a median permutation of $\mathcal{A}$ will be $i \prec j$. In that case, we can add $w(e_{ji})$ to the lower bound (since all permutations with $j \prec i$ in $\mathcal{A}$ disagree with a median permutation) then set its value to $w(e_{ji}) = 0$ in G'.

At first glance, incorporating the constraints seems to be interesting but one will quickly observe that the constraints previously found by the MOT method are only of the type $C_{ij} = 1$ where $i \prec j$ is the major order, adding nothing to the lower bound because in the graph G', the associated minor order edge e'_{ji} has weight $w(e'_{ji}) = 0$, by construction.

Nevertheless, the superiority of calculating a lower bound with constraints will appear in Section 4.4, when combined with an upper bound.

For $\mathcal{A} \subseteq \mathbb{S}_n$, we will denote the lower bound with set of constraints C by $Lb_{\mathcal{A}}(C)$. If $C = \emptyset$, then $Lb_{\mathcal{A}}(\emptyset)$ becomes $LowerBound_2$ associated with 3-cycles, described above.

4.3. An upper bound

In Section 3, we detailed a simulated annealing heuristic that finds a approximative solution for our problem. The approximative solution is a valid permutation therefore its distance to $\mathcal{A}$ will be used as an upper bound. We will denote this upper bound by $Ub_{\mathcal{A}}$.

Naturally, if $Lb_{\mathcal{A}}(C) = Ub_{\mathcal{A}}$ for a particular instance of the problem and any set of valid constraints C, then the problem is solved: the median distance is $Ub_{\mathcal{A}}$ and the associated permutation to that upper bound will be a solution *i.e.* a median of $\mathcal{A}$.

4.4. Putting everything together

Given a set of permutations $\mathcal{A} \subseteq \mathbb{S}_n$, we can deduce a valid set of constraints C using the MOT methods described in Section 4.1 and then apply the technique described in Section 4.2 to obtain the lower bound $Lb_{\mathcal{A}}(C)$. Running the SA heuristic of Section 3 will get us the upper bound $Ub_{\mathcal{A}}$.

We can use these upper and lower bounds to search for new constraints simply by adding a new possible constraint and verifying if the lower bound has exceeded the upper bound with this new add on. To do so, let us choose a pair of elements (i, j) for which the ordering is still unknown in the median, *i.e.* for

which $C_{ij} = 0$ and $C_{ji} = 0$. Now, let us suppose that $i \prec j$ in a median of $\mathcal{A}$ and let C' be the set of constraints C augmented with this new constraint $C_{ij} = 1$. If $Lb_{\mathcal{A}}(C') > Ub_{\mathcal{A}}$ then the added constraint $i \prec j$ is false, which means that $j \prec i$ in a median of $\mathcal{A}$ and we can add $C_{ji} = 1$ to our set of constraints C.

As finding a new constraint give an advantage to find others (as there is more knowledge about the structure of the optimal solution), we can redo the same process including unknown constraints that previously failed the test, repeating until no new constraint is found. We will called this new way of finding constraints the LUBC (lower-upper bounds and constraints) method.

Without surprise, as MOT3 method also benefits from new constraints, we propose a method that will iteratively alternate between MOT and LUBC until both are unable to find any additional constraint and call it MOT+LUBC.

As seen before, the constraints found by the MOT method are only of the type $C_{ij} = 1$ where $i \prec j$ is the major order. An advantage of the LUBC is the possibility to find constraints of $\mathcal{A}$ that are of the minor type i.e. where $C_{ij} = 1$ and $i \prec j$ is the minor order ($R_{ij} > R_{ji}$). Recalling the construction of G', the weight of an edge associated with the major order is strictly positive and leads to a non-zero augmentation of the lower bound. When the LUBC methods finds any new valid constraint of the minor order type, the augmentation of the lower bound is advantageous to find further new contraints.

The great efficiency of this new method can be observed in Table 11, where we tested our different approaches on distributed random sets of m permutations of $[n]$, with $m \in [3; 50]$ and $n = 15, 20, 30$ or 45. On our instances, the gain in the number of constraints found is ranging from +1% to 41% passing from MOT (Table 10*a*) to MOT+LUBC (Table 11*a*) and from +0.1% to 36% passing from MOTe (Table 10*b*) to MOTe+LUBC (Table 11*b*). We can note that all instances of $n \leq 20$ have a average resolution rate higher than 90% with the MOTe+LUBC method.

Although the method seems very efficient in average, we observed a big variance between instances. We did a small study in Table 12 to calculate the range of the standard deviation on the percentage of solved pairs for each size n of permutations. For exemple, the MOTe+LUBC will solve in average 70% pairs of elements for a $m = 20$, $n = 30$ instance but, with a standard deviation of 21%, it would be very normal to solve between 49% and 91% pairs of elements on another instance with the same m and n. This can have a big impact on solvers which rely on those constraints. In general, the standard deviation is smaller for small n, when the efficiency was very high.

4.5. Pseudocode and time complexity of the new MOT+LUBC and MOTe+LUBC approaches

Algorithm 2 and Algorithm 3 give the pseudocode of our MOT+LUBC method. For the pseudocode of MOTe+LUBC, simply replace $\text{MOT}(\mathcal{A}, C, i, j)$ in Algorithm 2 by $\text{MOT}e(\mathcal{A}, C, i, j)$.

Now for the time complexity, as the complexity of the MOT method is of $O(n^2m + n^3 + n^2k)$ (see [17]), the complexity of the LUBC method is $O(n^2m +$

a)

m/n	10	15	20	25	30	35	40	45
3	0.933	0.921	0.885	0.817	0.725	0.626	0.560	0.513
5	0.928	0.913	0.859	0.746	0.605	0.495	0.428	0.381
10	0.842	0.845	0.809	0.729	0.616	0.498	0.409	0.347
15	0.948	0.939	0.890	0.755	0.569	0.436	0.368	0.328
20	0.888	0.884	0.843	0.746	0.597	0.462	0.377	0.327
25	0.958	0.949	0.903	0.771	0.579	0.432	0.358	0.322
30	0.907	0.904	0.861	0.755	0.589	0.448	0.364	0.317
35	0.964	0.955	0.909	0.778	0.582	0.435	0.359	0.320
40	0.919	0.915	0.873	0.759	0.590	0.446	0.362	0.323
45	0.968	0.959	0.913	0.784	0.586	0.433	0.363	0.317
50	0.928	0.923	0.881	0.766	0.589	0.441	0.358	0.314

b)

m/n	10	15	20	25	30	35	40	45
3	0.977	0.966	0.952	0.924	0.873	0.809	0.727	0.669
5	0.964	0.954	0.929	0.867	0.766	0.654	0.567	0.516
10	0.972	0.951	0.915	0.848	0.747	0.635	0.528	0.454
15	0.960	0.953	0.919	0.823	0.668	0.525	0.446	0.398
20	0.962	0.945	0.908	0.824	0.697	0.554	0.455	0.399
25	0.965	0.958	0.923	0.820	0.653	0.504	0.421	0.375
30	0.962	0.947	0.910	0.820	0.674	0.529	0.429	0.379
35	0.969	0.961	0.926	0.819	0.642	0.486	0.406	0.356
40	0.963	0.950	0.912	0.815	0.660	0.509	0.420	0.371
45	0.972	0.964	0.927	0.818	0.638	0.486	0.401	0.363
50	0.965	0.952	0.914	0.816	0.654	0.506	0.416	0.366

c)

n	10	15	20	25	30	35	40	45
nbInstances	100000	80000	50000	20000	10000	4000	2000	1000

Table 11: Average efficiency of *a*) MOT+LUBC and *b*) MOTe+LUBC constraints in % of resolution of pairs of elements on sets of uniformly distributed random sets of m permutations, $m = 3$ and $m = 5x$, $1 \leq x \leq 10$ of $[n]$, $n = 5x$, $2 \leq x \leq 9$. The table in *c*) gives the number of sets generated for each n.

method/n	10	15	20	25	30	35	40	45
MOT+LUBC	0.05-0.10	0.04-0.09	0.08-0.12	0.12-0.21	0.18-0.27	0.15-0.25	0.09-0.22	0.06-0.16
MOTe+LUBC	0.03-0.07	0.03-0.07	0.06-0.10	0.09-0.20	0.10-0.24	0.15-0.25	0.09-0.20	0.07-0.17
instances	100	100	100	100	100	50	50	50

Table 12: Range (min-max) for values of standard deviation for MOT+LUBC and MOTe+LUBC methods. We generated 50-100 random uniform instances for each couple (m, n), calculated the constraints and noted the standard deviation. For each n we noted the range (min-max) of the standard deviation values. This table gives a small insight on our search space reduction methods efficiency variation.

$n^3 + n^5k$), since the lower bound method in $O(n^3)$ is repeated on all pairs of elements over k iterations (number of times we find new constraints). We observed in practice (see Figure 3) that the time required to compute the new constraints method is amortized in $O(n^3)$. We also observed that the average number of iterations for any sets of m permutations of $[n]$ ranges in $k \in [2.4, 4.4]$ for MOT, in $k \in [3.3, 11.2]$ for MOT+LUBC, in $k \in [2.9, 7.4]$ for MOTe and in $k \in [3.1, 13.0]$ for MOTe+LUBC.

Polynomial regression for the MOT+LUBC gives us an approximation of the

Algorithm 2: MOT_LUBC_constraints($\mathcal{A}$, UpperBound)

```
Data: a set 𝒜 of permutations, an known upper bound UpperBound
Result: a set of valid constraints
begin
    C ← ∅      // set of constraints found
    improved ← True      // If true, continue to look for new constraints
    while improved do
        improved ← False
        for i ← 1 to n do
            for j ← i + 1 to n do
                if MOT(𝒜, C, i, j) then
                    C ← C ∪ {(i, j)}     // (i,j) new order constraint found by MOT
                                            theorems (described in Section 4)
                    improved ← True      // Continue to look for new constraints
                end if
            end for
        end for
        if !improved then
            for i ← 1 to n do
                for j ← i + 1 to n do
                    if LUBC(𝒜, C, i, j, UpperBound) then
                        C ← C ∪ {(i, j)}     // if no more constraints found by MOT,
                                                we try LUBC
                        improved ← True
                    end if
                end for
            end for
        end if
    end while
    return C
end
```

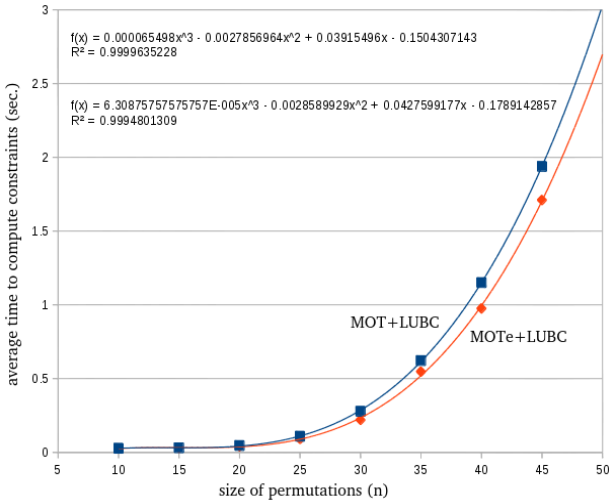


Figure 3: Average time in seconds to compute the new constraints methods on one instance of (m, n) with $m \in \{3, ..., 50\}$ for a specific n. In blue, the MOT+LUBC time and in orange, the MOTe+LUBC time. Average calculated over 1000-100000 instances per (m, n).

Algorithm 3: LUBC($(\mathcal{A}, C, i, j, \text{UpperBound})$)

```
Data: a set A of permutations, a set C of valid constraints, first element i, second element
      j, an known upper bound UpperBound
Result: If i ≺ j is a valid constraint
begin
    isValid ← False
    n ⟵ number of elements in the permutations of A
    LowerBound ← 0
    C' ← C ∪ (j, i)      // add the reverse of the new constraint to test in the set of
                            constraints
    C' ← transitive_closure(C')      // apply transitivity rule over the new set of
                                        constraints
    V = {1, 2, 3, ..., n − 1, n}
    E = {(i, j)|i ∈ V, j ∈ V, i ≠ j, i ≺ j is a major order}
    G = (E, V)      // create majority graph
    for (i, j) ∈ E do
        w(i, j) = R_ji − min{R_ij, R_ji}
        LowerBound ← LowerBound + min{R_ij, R_ji}      // compute the trivial lower
                                                          bound
    end for
    for (i, j) ∈ C' do
        LowerBound ← LowerBound + w(j, i)      // add known orders costs to the
                                                  lower bound
        w(j, i) = 0
    end for
    for i ← 1 to n      // go throughout all cycles of length 3 in the graph do
        for j ← 1 to n do
            for k ← 1 to n do
                min ← min{w(i, j), w(j, k), w(k, i)}
                if min > 0 then
                    w(i, j) ← w(i, j) − min
                    w(j, k) ← w(j, k) − min
                    w(k, i) ← w(k, i) − min
                    LowerBound ← LowerBound + min      // add minimal cost of
                                                          cycles of length 3
                                                          found in the graph
                end if
            end for
        end for
    end for
    if LowerBound > UpperBound then
        isValid ← True      // if the reversed new constraint yields a higher lower
                               bound than the best upper bound, then the new
                               contraint is valid
    end if
    return isValid
end
```

time (in seconds): $\text{time}(n) = 0.0000655n^3 - 0.002786n^2 + 0.03915n - 0.15043$. We can extrapolate that it will take around 3 seconds in average to calculate the new constraints (MOT+LUBC) on a random instances of (m, n), with $n = 50$ and $m \leq 50$.

4.6. Experimental work with our space reduction technique

We tested our space reduction technique on the random uniform data sets and on the real life data sets presented in Section 2.2. Results of tests of MOTe+LUBC and MOT+LUBC on random uniform data sets can be found in Table 11 in Section 4.4. When tested on real data sets from PrefLib [20],

MOTe+LUBC gave excellent results. MOTe+LUBC ordered all pairs (100%) of elements on all small instances ($n < 5$). For all medium instances ($5 \leq n < 60$), MOTe+LUBC ordered at least 98% of all pairs of elements, but, most of the time, it ordered all pairs. For the big instances ($60 \leq n < 170$), MOTe+LUBC usually ordered between 96% and 100%

of pairs of elements with one instance at 83%. Even if LUBC is more time consuming on big instances, it is still worth using it. On the "ED-00015-00000032.soc" instance ($m = 4, m = 153$), MOTe ordered 76.17% of the pairs but adding LUBC, it raised up to 99.97% of solved pairs. For the giant instances, LUBC had to be turned off because of time efficiency. We noticed that from the 6 giant instances, two were duplicates. The MOTe ordered 93.71%, 94.94%, 50.73% and 60.33% pairs of elements on the four giant instances. On the other hand, MOT+LUBC was less efficient than MOTe on all data sets. We recommend therefore to use MOTe+LUBC if a complete enumeration of all of the medians is not necessary.

4.7. Combining with another space reduction technique

In [31], Truchon proposed a space reduction technique called the Extended Condorcet Criterion (ECC) which, as indicated by its name, is based on the Condorcet Criterion [32]. In this last criterion, an element that is preferred pairwise with all other elements will be the first element in any median permutation and is called the Condorcet winner. Extending this criterion, the ECC states that if the set of elements $\{1, ..., n\}$ can be partitioned into two subsets X and Y such that $x \prec y$ is a major order $\forall x \in X$ and $\forall y \in Y$ then, all elements of X will be placed to the left of all elements of Y in any median permutation. A weaker version, that outputs more constraints but that only work on a subset of medians, is calculated as follows: construct a strict majority graph where each node corresponds to an element and arcs correspong to major order relations between pairs of elements, then calculate the strongly connected components and their topological ordering. Some median permutations will respect the element ordering of the topological order of those components. We will only consider the weak version of ECC in this work, as we prefer MOTe+LUBC to MOT+LUBC, aiming to find only a median and not to enumerate them. In [13], authors showed that ECC (weak version) is subsuming their previous space reduction technique (called the 3/4-Majority Rule) and combined ECC with their solver obtainaing impressing results, specially on real life data sets.

We compared and combined our MOTe+LUBC method with the ECC method on the random uniform data sets and real life data sets of Section 2.2. We have a few interesting remarks regarding those two approaches. A first remark is that neither MOTe+LUBC nor ECC is subsumed by the other. Thus combining them is of interest.

Second, MOTe+LUBC and ECC can yield contradicting constraints. This is not surprising as both methods are giving constraints valid only on a subset of medians. An example is found in the PrefLib instance "ED-00015-00000011.soc". A simplified example is the set $\mathcal{A} = \{[4, 5, 2, 3, 1], [1, 3, 4, 5, 2],$

$[5, 1, 2, 3, 4]$, $[1, 2, 3, 4, 5]\}$. In this example, we have strict majorities for (ordered) pairs $(1, 2)$, $(1, 3)$, $(1, 4)$, $(2, 3)$, $(3, 4)$, $(4, 5)$ and $(5, 2)$. All other pairs are at equality. In the ECC point of view, there are 2 strongly connected components: $\{1\}$ and $\{2, 3, 4, 5\}$, thus 1 will precede all the other elements in a median. But note that elements $\{2, 3, 4, 5\}$ are forming a 4-cycle $(2, 3, 4, 5)$ and the crossing pairs $\{2, 4\}$ and $\{3, 5\}$ are at equality thus any ordering of elements following this cycle is valid in a median, including $[5, 2, 3, 4]$. Then one median of $\mathcal{A}$ is $[1, 5, 2, 3, 4]$. But as the pair $\{1, 5\}$ is at equality (there is no majority preference over them), we can swap them to get $[5, 1, 2, 3, 4]$, which is another median of $\mathcal{A}$. In our case, the MOTe method found the $5 \prec 1$ constraint which contradicts the $1 \prec 5$ constraint found by ECC.

Third, in our experiments, MOTe+LUBC generally yields more constraints on pairs of elements than ECC. In random uniform data sets, MOTe+LUBC is greatly outperforming ECC, which barely gives constraints. We assume that ECC is not well suited for random data because of the lack of consensus structure as, in contrast, the MOTe+LUBC is a element pairwise approach that does not require consensus but only some conditions when a pair of elements is often in the same order in the permutation set. In real life data sets from PrefLib, ECC was very effective. It ordered 100% of the pairs for all small instances. On medium and big instances, it often ordered between 80 to 100% of the pairs. But, for each instance, MOTe+LUBC always ordered more pairs.

Fourth, we combined ECC and MOTe+LUBC together and evaluated their combined efficiency. As both methods can yield contradicting constraints, combining them has to be done in a careful way. In our implementation, we first compute ECC and then MOTe+LUBC. The results were not very impressing, the combined method being very similar to MOTe+LUBC alone. On all small, medium and big instances of the real life data sets, the percentage of ordered pairs of elements was the same as MOTe+LUBC except for 3 instances: two whose gain, in terms of new ordered pairs, were very small (+0.11% and +0.30%) and one whose gain was more significant (+4.25%, passing from 83.96% to 88.21%). We also did not noticed any improvement on the computation time for both synthetic and real life data. As the LUBC method is more time consuming than MOTe, we assume that it is also the case towards ECC.

Even thought integrating ECC in our approach does not speed up our method and that the gain in new constraints is very small, we still advise to combine this method because it sometimes permits to gain new constraints. We refer to this combined method as ECC+MOTe+LUBC.

5. Two exact approaches

To solve the problem exactly, few approaches have already been looked at: branch-and-bound ([13, 16, 18, 33]), dynamic programming ([13]) and integer linear programming ([13, 15, 18, 19]). The next subsections present our work in those directions.

5.1. Branch-and-bound

Our third approach is an exact branch-and-bound solver for the problem that combines the result of the simulated annealing method of Section 3 with the constraints obtained in Section 4.4 to avoid exploring not promising search subtrees.

Our branch-and-bound algorithm simply constructs possible medians of $\mathcal{A} \subseteq \mathbb{S}_n$, *i.e.* permutations of $[n]$ by putting a new element to the right of the already known ones until no more element are available. Thus we explore a tree having the empty permutation at its root, permutations of any k elements of $[n]$ as nodes of level k and where the leaves are permutations of $\mathbb{S}_n$. Each node N will have a corresponding lower bound $Lb(N)$ representing the fact that for all permutations π derived from N, $d_{KT}(\pi, \mathcal{A}) \geq Lb(N)$.

If this lower bound $Lb(N)$ is higher than the current upper bound of the problem, then all the permutations derived from it will have a higher distance than one already found and node N with all its descendants can be omitted in the exploration. As the number of nodes/leaves is finite and the bounding method only cuts branches that do not represent possible medians, the BnB will always converge to the optimal solution.

More specifically, given our set $\mathcal{A} \subseteq \mathbb{S}_n$, we run the SA heuristic of Section 3 to have a first approximative median of $\mathcal{A}$, π_{approx}, and a first upper bound $Ub_{\mathcal{A}}$, which will always represent the score associated with the currently best solution found. The approximative solution will serve as guidance so that the first leaf that will be visited by the BnB will be π_{approx}. This guarantees us an efficient cutting of non-promising branches.

A node at level k is represented by a vector of size k, $x = [x_1, ..., x_k]$ which corresponds to a permutation in construction. Let S be the set of elements still to be placed *i.e* $S = [n] - \{x_1, x_2, ..., x_{k-1}, x_k\}$ and L a list which orders S. At the beginning of our BnB, $S = [n]$ and $L = \pi_{\text{approx}}$. The BnB will branch by choosing the next element from the list: $\ell_i \in L$ that will be placed at the immediate right of x_k. We are going to apply the bound and cuts Bound_1, Cut_1, Cut_2 and Cut_3 described below for each choice of ℓ_i. If it succeeds passing all the bounds test, $x_{k+1} := \ell_i$, and we go down to the new node $x' = [x_1, x_2, ..., x_{k-1}, x_k, x_{k+1}]$. If not, we try ℓ_{i+1} as a possible x_{k+1}. If we go down to a leaf, *i.e.* if $k+1 = n$, its corresponding permutation π_{leaf} is compared to the best solution. If $d_{KT}(\pi_{\text{leaf}}, \mathcal{A}) = Ub_{\mathcal{A}}$, we add π_{leaf} to the current set of medians. If $d_{KT}(\pi_{\text{leaf}}, \mathcal{A}) < Ub_{\mathcal{A}}$, then it is our new upper bound and we change our set of medians so that it contain only this new optimal permutation π_{leaf}. The BnB backtracks after all possibilities of $\ell_i \in L$ for x_{k+1} had been explored or after investigating a leaf node.

Bounds and Cuts. Now, let us set everything that is needed to described our different bound and cuts. First, given a set of permutations $\mathcal{A} \subseteq \mathbb{S}_n$, we deduce a valid set of constraints $C(\mathcal{A})$ using the method MOTe+LUBC described in Section 4.4. We also pre-calculated, for each triplet of elements x, y and $z \in [n]$ the best ways to put them all together, one after the other, in a

median of $\mathcal{A}$. All the other ways to order them consecutively in a permutation will be called a forbidden triplet, since a permutation containing this ordering cannot be a median of $\mathcal{A}$. As storing forbidden triplets takes $O(n^3)$ space, we did not considered k-tuples, with $k \geq 4$, which would take $O(n^k)$ space and would be less scalable.

For each node $x = [x_1, ..., x_k]$, we can compute a distance $d(x, \mathcal{A})$ in the following way:

$$d(x, \mathcal{A}) = \underbrace{\sum_{i=1}^{|x|} \sum_{j=i+1}^{|x|} R_{x_i x_j}(\mathcal{A})}_{\substack{\text{contribution to the Kendall-}\tau \\ \text{distance for the elements} \\ \text{already placed}}} + \underbrace{\sum_{i=1}^{|x|} \sum_{j=1}^{|L|} R_{x_i l_j}(\mathcal{A})}_{\substack{\text{contribution obtained by the fact} \\ \text{that all elements of } x \text{ are to the} \\ \text{left of elements in } L \text{ (yet to be placed)}}}.$$

Finally, for each node x, let $b(x)$ be the boolean vector of length n representing which elements of $[n]$ are already placed in this node (*i.e* we have $b_i(x) = 1$, $1 \leq i \leq n$, if and only if $i = x_j$, $1 \leq j \leq k$). So if a node x' contained the same k elements of node x but in a different order, $b(x) = b(x')$. Our BnB will construct and update a set TopScores of pairs $< b, v >$, where b is any boolean vector of length n and $v = \min d(x, \mathcal{A})$, for x already explored such that $b(x) = b$.

Now, if our current best solution is π_{best}, our current upper bound is $Ub_{\mathcal{A}} = d_{KT}(\pi_{best}, \mathcal{A})$, our current node is $x = [x_1, ..., x_k]$, L is an ordered list of the elements still to be placed, and we are studying $\ell_i \in L$ as a possible x_{k+1}, we have that:

- **Cut$_1$**: If (x_{k-1}, x_k, ℓ_i) is a forbidden triplet then x_{k+1} cannot be l_i and so it is rejected. (*i.e* we do not explore the subtree having node $x = [x_1, ..., x_k, \ell_i]$ as its root.)

- **Cut$_2$**: If there exist $\ell_j \in L$, $\ell_j \neq \ell_i$ such that $C(\mathcal{A})(\ell_j, \ell_i) = 1$ then we know that ℓ_i has to be to the right of ℓ_j in a median permutation of $\mathcal{A}$ and so x_{k+1} cannot be ℓ_i and is rejected.

- **Cut$_3$**: Let $x' = [x_1, \ldots, x_k, \ell_i]$. If $< b(x'), v > \in$ TopScores and $d(x', \mathcal{A}) > v$ then x_{k+1} cannot be ℓ_i and so it is rejected.

- **Bound$_1$**: Let a lower bound of a list L be

$$lb(L) = \sum_{i=1}^{|L|} \sum_{j=i+1}^{|L|} (\min\{R_{l_i l_j}, R_{l_i l_j}\}) + tri(L),$$

  where $tri(L)$ is a simpler implementation of the lower bound using only 3-cycles described in Section 4.2. In this implementation, the 3-cycles are pre-calculated at the beginning, and the contribution of a cycle is added if and only if all three of its elements are in L. Let $x' = [x_1, \ldots, x_k, \ell_i]$

and let L' be the list L without ℓ_i. Let $Lb(x') = d(x', \mathcal{A}) + lb(L')$. If $Lb(x') > Ub_{\mathcal{A}}$ then x_{k+1} cannot be ℓ_i and so it is rejected.

Algorithm 4 gives the pseudocode of our BnB method.

Our BnB can solve in reasonable time (a few seconds in average) any problem with $n \leq 38, m \leq 50$, since we kept all calculation in linear time at the node level for efficiency purpose. The case where $m = 4$ is the hardest case (and the most variable in execution time) for the BnB, opposite to SA, as the average number of medians of $\mathcal{A} \subseteq \mathbb{S}_n$ have been observed to be the biggest for all m.

Note that when some features are turned off (SA guidance, MOTe+LUBC constraints, forbidden triplets, TopScores data), our BnB algorithm ressemble a lot the BnB of [34] where nodes are corresponding to prefixes of permutations and the lower bound of [15] is used to cut not promising subtrees. This BnB was not tested over $n \geq 15$. In [18], Ali and Meilă used their BnB on larger n but as an approximation algorithm by limiting the available memory and by limiting the number of explored nodes.

5.2. Linear Programming with CPLEX

Our forth and last approach is an integer linear programming formulation of the problem. We can model the problem of the median of a set $\mathcal{A}$ of permutations in integer linear programming (ILP) using the standard formulation of the linear scheduling problem (LOP):

$$minimize : \sum_{i<j,\ i,j\in\{1,\dots,n\}} R_{ij}(\mathcal{A}) x_{ij}$$

subject to:

$$x_{ij} + x_{ji} = 1, \quad i \neq j \in \{1, ..., n\},$$

$$x_{ij} + x_{jk} + x_{ki} \geq 1, \quad i \neq j \neq k \in \{1, ..., n\},$$

$$x_{ij} \in \{0, 1\} \quad i \neq j \in \{1, ..., n\},$$

The variables x_{ij} define the order between the elements i and j in a median permutation: if $x_{ij} = 1$ then i is to the left of j ($i \prec j$). In the cost function, R_{ij} indicates the number of permutations in $\mathcal{A}$ in which i is right of j (see Definition 1). The constraint $x_{ij} + x_{ji} = 1$ imply that only one order can exist between the two elements i and j, either $i \prec j$ or $j \prec i$. The constraint $x_{ij} + x_{jk} + x_{ki} \geq 1$ imposes the transitivity of the $\prec$ relation: if $i \prec j$ and $j \prec k$, then $i \prec k$. Finally, constraint $x_{ij} \in \{0, 1\}$ bounds the variables.

The resolution of this IP problem gives a score, which is the Kendall-τ distance between a median and the set of permutations $\mathcal{A}$ and boolean values for the variables $x_{ij}, i \neq j \in \{1, ..., n\}$ that can be converted into permutation. If we relax the last constraint (relaxing the integrality), we get an interesting lower bound to the problem by relaxation into linear programming.

We implemented an IBM CPLEX solver in Java (using Concert Technology) to solve the problem of the median of permutations in this ILP model. This solver is the best one available for this problem, up to now, according to [18].

Algorithm 4: Branch-and-bound($\mathcal{A}, x, L$)

Data: a set $\mathcal{A}$ of permutations, the vector x representing the permutation in construction, L the list of elements to set
Result: The set M containing the medians permutations
begin
  $n \longleftarrow$ number of elements in the permutations of $\mathcal{A}$
  $k \longleftarrow$ number of elements in the vector x
  if $L = \emptyset$ `// if the node is a leaf (complete permutation)` **then**
    $\pi \leftarrow$ permutation associated to x
    if $d_{KT}(\pi, \mathcal{A}) <$ UpperBound `// if the permutation is the best so far` **then**
      $M \leftarrow \{\pi\}$
      UpperBound $\leftarrow d_{KT}(\pi, \mathcal{A})$
    else if $d_{KT}(\pi, \mathcal{A}) =$ UpperBound **then**
      `// if not, just add the permutation to the solution pool`
      $M \leftarrow M \cup \{\pi\}$
    end if
  else
    for $i \leftarrow 1$ **to** $|L|$ **do**
      $x_{k+1} \longleftarrow l_i$ `// test each element from L as the next element`
      isAccepted $\leftarrow$ True
      $x' \leftarrow (x_1 x_2, \ldots, x_k, x_{k+1}, 0, \ldots, 0)$ `// create corresponding (k+1)-permutation`
      $L' \leftarrow L \setminus \{x_{k+1}\}$
      $b(x') \leftarrow b(x)$
      $b(x')_{x_{k+1}} \leftarrow 1$ `// create corresponding binary vector`
      $kd(x') = \sum_{i=1}^{|x'|} \sum_{j=i+1}^{|x'|} R_{x'_i x'_j} + \sum_{i=1}^{|x'|} \sum_{j=1}^{|L'|} R_{x'_i l'_j}$ `// compute partial score`
      $lb(L') = \sum_{i=1}^{|L'|} \sum_{j=i+1}^{|L'|} (\min\{R_{l'_i l'_j}, R_{l'_i l'_j}\}) + tri(L')$ `// compute lower bound`
      if $x_k \prec x_{k+1}$ *is the minor order* **then**
        isAccepted $\leftarrow$ False `// test neighbour optimality`
      end if
      if $x_{k-1} \prec x_k \prec x_{k+1}$ *is not optimal* **then**
        isAccepted $\leftarrow$ False `// test forbidden triplets`
      end if
      if $\exists l_i \in L' \mid C_{l_i, x_{k+1}} = 1$ **then**
        isAccepted $\leftarrow$ False `// test constraints`
      end if
      if $kd(x') + lb(L') >$ UpperBound **then**
        isAccepted $\leftarrow$ False `// test lower vs upper bound`
      end if
      if $< b(x'), v >\in$ TopScores *(for any v)* **then**
        if $kd(x') > v$ **then**
          isAccepted $\leftarrow False$ `// test if the elements were already better placed using TopScores set`
        else if $kd(x') < v$ **then**
          TopScores $\leftarrow$ TopScores $\setminus < b(x'), v >$
          TopScores $\leftarrow$ TopScores $\cup \{< b(x'), kd(x') >\}$ `// update TopScores set with better element`
        end if
      else
        TopScores $\leftarrow$ TopScores $\cup \{< b(x'), kd(x') >\}$ `// add new entry in TopScores set`
      end if
      if isAccepted **then**
        Branch-and-bound($\mathcal{A}, x', L'$) `// branch with new element`
      end if
    end for
  end if
end

The CPLEX solver allows to use a given permutation as a starting point to maybe help the computation. It also allows to add constraints for all the pairs of elements $(i, j), i \neq j \in \{1, ..., n\}$ for which we already know their relative order in the median (i.e. if $i \prec j$ or $j \prec i$). We decided to investigate if either using the solution of our simulated annealing algorithm (see Section 3) as a starting point for the solver or integrating the constraints found in Section 4 or both would make the problem easier to solve in term of computation time. Table 13 presents the result of our quick investigation. Note that in this case, we only test on one instance per (m, n) pair to show individual variations between the four different combination of approaches.

As you can see, the introduction of the simulated annealing solution or the constraints is not automatically linked to an increase in performance of the solver. On the other hand, when we do have an increase of performance it is significant and for those few cases where we do not, the time difference between using the solver as it is or with the added information is negligible. What we need to investigate in more detail is when it is profitable to use the constraints, since it seems to really depend on the instance of the problem considered as shown in Table 13. Note that the time to run the SA heuristic is negligible in practice compared to the runtime of CPLEX. The time to calculate the constraints is greater on small instances but the trend reverses completely for larger sizes to the point where it represent only a small fraction of the running time of CPLEX.

5.3. Experimental work with the branch-and-bound and ILP approaches

Our branch-and-bound approach can solve in reasonable time random uniform small to medium instances up to $n = 38$. Its efficiency is even greater on real data sets: using MOTe+LUBC constraints and π_{approx} obtained from our SA heuristic as guidance, it was able to solve all real life instances up to $n = 95$.

We also did some quick comparing of our BnB approach with the recent Betzler *et al.* [13] branch-and-bound and dynamic programming approches on the real data sets of Section 2.2. Note that all the approaches are combined approaches, as our branch-and-bound benefits from SA and MOTe+LUBC and the two compared approaches benefits from their ECC search space reduction technique, therefore performance is greatly affected by the preprocessing. It resulted in no clear winner, the two following examples supporting this claim and giving the general tendency. For the "ED-00015-00000010.soc" instance ($m = 4$,$n = 71$), our branch-and-bound solved it after 12 seconds but neither of the two compared approaches was able to solve it in the allowed 20 minutes. On the other hand, for the "ED-00015-00000011.soc" instance ($m = 4$,$n = 63$), both compared approaches clearly outperformed our approach by solving it in a fraction of a second as our method took over 4 seconds. Thus we claim that our method is competing with the best free solvers.

The ILP approach using the commercial solver CPLEX, which is consider to be the best solver for this problem [13, 18, 19], gave also the best results in our work. This approach is clearly stronger than our branch-and-bound but is relying on a commercial ILP solver. When we integrated the space reduction

	$m = 3$			
n	IP	IP + SA	IP + C	IP + SA + C
60	515.56	646.21	449.62	579.77
70	1563.79	1788.11	1396.94	1606.77
80	304903.18	143389.36	106089.00	77493.30
90	4140.55	4586.23	3475.34	3907.65
100	775017.20	104621.75	160247.34	37931.90

	$m = 4$			
n	IP	IP + SA	IP + C	IP + SA + C
60	454.89	578.87	203.38	332.12
70	720.38	909.04	116.75	332.31
80	1470.65	1771.20	1209.36	1517.77
90	197793.46	15106.97	132881.12	115424.61
100	4182.20	4807.44	5100.42	5630.65

	$m = 5$			
n	IP	IP + SA	IP + C	IP + SA + C
60	14288.28	8788.22	9688.74	6183.98
70	1931.09	2134.26	2063.31	2270.31
80	33023.36	4540.35	26113.12	4975.42
90	2781696.72	2383961.55	679621.16	908664.74
100	136065.61	14990.22	101878.41	15293.40

Table 13: Resolution time (in CPLEX ticks) for solving instances of the median problem for sets of m = 3,4,5 permutations and size ranging from n= 60 to n = 100. Each pair n and m represents an unique set of permutations that was generated randomly (uniformly) using Fischer-Yates shuffle. For each instance the resolution time is given for: the standard model (ILP), ILP + starting point provided by the heuristic SA (ILP+SA), ILP with added constraints (ILP+C) and the model integrating everything (ILP+SA+C). The colored cells indicated the best options to solve each instance.

constraints from MOTe+LUBC to our model and the SA heuristic that provided a warm start to the solver, we were able to solve any random uniform instance up to $n = 70, m = 50$ and $n = 80, m = 10$ in reasonable time (less than 2 minutes). We were also able to solve all real life instances including the "ED-00015-00000004.soc" instance which, to our knowledge, was never solved before. This specific instance of $m = 5$ permutations of length $n = 242$ took 8 min 3 sec (67k ticks) on 1 thread to solve and gave the optimal Kemeny score of 33497. We estimated the Mallows dispersion parameter θ to be 0.0265. As a note, the MOTe constraints were able to order 60.33% of the pairs of elements of this example. We did not run the LUBC constraints due to time efficiency. In [13], the authors could only order less than 28% of the pairs of elements and were not able to calculate the median for that instance. Note that they had a 5 minutes threshold upon which calculations were stoppped.

More investigation has to be done but our experiments seems to show that a combination of methods (ILP solver, SA and ECC+MOTe+LUBC) would give an optimal way, in average, to solve the problem of the median of permutations.

6. Conclusion

In this article, we studied the problem of finding the median of a set of m permutations $\mathcal{A} \subseteq \mathbb{S}_n$ under the Kendall-τ distance. This problem is known to be NP-complete for $m \geq 4$, m even. This work presents four different solving techniques for this problem. First a well parameterized simulated annealing heuristic is presented with each parameter choice supported with extensive testing. This heuristic was designed to give a close-to-optimal solution with a great probability and in reasonable time. We showed that this heuristic can outperform other state-of-the-art heuristics. Second, comes a space reduction technique based on finding constraints on relative orders between elements. Those constraints are found by calculating a lower bound on a subspace of all permutations. This space reduction seems stronger that previously known space reduction techniques. Third, a branch-and-bound solver is described in detail where cuts include local optimality and lower bounding. The usage of memory permits to avoid recalculations of similar subtrees. This algorithm is a good free alternative to solve real life instances of the problem. Finally, an integer linear programming model is formulated and solved by CPLEX. The BnB algorithm and the ILP CPLEX approach are powered by the first and second solving techniques. The simulated annealing solution give guidance and an advance start for exact algorithms as the space reduction technique significantly reduces the solving time. Using the ILP approach combined with our previous approaches, we could solve the "ED-00015-00000004.soc" instance from PrefLib, for the first time in our knowledge.

Ideas for future works includes an extensive comparison with other free exact solvers and heuristics, as well as testing on various synthetic data sets. We still have a few ideas to explore to improve our approaches based on other works: in [28], authors are combining a local search heuristic using circular moves with a local exact solver that refines a segment of the permutation and in [13], authors make use of parallelization in their algorithms. Both those ideas are worth investigating as they already yield good results. Furthermore, it would also be interesting to take into account the fact that in real life, the rankings considered are not always permutations: sometimes elements are missing and further, ranking schemes can often rank several elements in the same position, so rank ties should be considered.

7. Acknowledgements

We would like to thanks our anonymous reviewers of the conference version of this article for their careful and inspiring comments. We would also like to thanks our anonymous reviewers of this extended version for their helpful comments and for inspiring us to test our methods on big real life data sets which gave surprisingly good results.

References

[1] R. Milosz, S. Hamel, Heuristic, branch-and-bound solver and improved space reduction for the median of permutations problem, in: L. Brankovic, J. Ryan, W. Smyth (Eds.), Combinatorial Algorithms, Vol. 10765 of Lecture Notes in Computer Science, Springer International Publishing, 2018. doi:10.1007/978-3-319-78825-8_25.

[2] M. Kendall, A new measure of rank correlation, Biometrika 30 (1-2) (1938) 81–93. doi:10.1093/biomet/30.1-2.81.

[3] J. Kemeny, Mathematics without numbers, Daedalus 88 (4) (1959) 577–591.

[4] C. Dwork, R. Kumar, M. Naor, D. Sivakumar, Rank aggregation methods for the web, in: Proceedings of the 10th international conference on World Wide Web, WWW '01, ACM, 2001, pp. 613–622. doi:10.1145/371920.372165.

[5] T. Bield, F. Brandenburg, X. Deng, Crossings and permutations, in: P. Healy, N. Nokolov (Eds.), Graph Drawing, Vol. 3843 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2005, pp. 1–12. doi:10.1007/11618058_1.

[6] G. Bachmeier, F. Brandt, C. Geist, P. Harrenstein, K. Kardel, D. Peters, H. Seedig, k-majority digraphs and the hardness of voting with a constant number of voters, ArXiv e-printsarXiv:1704.06304.

[7] N. Ailon, M. Charikar, N. Newman, Aggregating inconsistent information: ranking and clustering, Journal of the ACM 55 (5) (2008) 1–27. doi:10.1145/1411509.1411513.

[8] C. Kenyon-Mathieu, W. Schudy, How to rank with few errors, in: Proceedings of the thirty-ninth annual ACM symposium on Theory of computing, STOC '07, ACM, 2007, pp. 95–103. doi:10.1145/1250790.1250806.

[9] A. van Zuylen, D. Williamson, Deterministic pivoting algorithms for constrained ranking and clustering problems, Mathematics of Operations Research 34 (3) (2009) 594–620. doi:10.1287/moor.1090.0385.

[10] N. Betzler, J. Guo, C. Komusiewicz, R. Niedermeier, Average parameterization and partial kernelization for computing medians, Journal of Computer and System Sciences 77 (4) (2011) 774–789. doi:10.1016/j.jcss.2010.07.005.

[11] M. Karpinski, W. Schudy, Faster algorithms for feedback arc set tournament, kemeny rank aggregation and betweenness tournament, in: K.-Y. C. O. Cheong, K. Park (Eds.), Algorithms and Computation, Vol. 6506 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2010, pp. 3–14. doi:10.1007/978-3-642-17517-6_3.

[12] N. Nishimura, N. Simjour, Parameterized enumeration of (locally-) optimal aggregations, in: R. S.-O. F. Dehne, J. Sack (Eds.), Algorithms and Data Structures, Vol. 8037 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2013, pp. 512–523. doi:10.1007/978-3-642-40104-6_44.

[13] N. Betzler, R. Bredereck, R. Niedermeier, Theoretical and empirical evaluation of data reduction for exact kemeny rank aggregation, Autonomous Agents and Multi-Agent Systems 28 (2014) 721–748. doi:10.1007/s10458-013-9236-y.

[14] G. Blin, M. Crochemore, S. Hamel, S. Vialette, Median of an odd number of permutations, Pure Mathematics and Applications 21 (2) (2011) 161–175.

[15] V. Conitzer, A. Davenport, J. Kalagnanam, Improved bounds for computing kemeny rankings, in: Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1, AAAI'06, AAAI Press, 2006, pp. 620–626.

[16] A. Davenport, J. Kalagnanam, A computational study of the kemeny rule for preference aggregation, in: Proceedings of the 19th National Conference on Artifical Intelligence, AAAI'04, AAAI Press, 2004, pp. 697–702.

[17] R. Milosz, S. Hamel, Space reduction constraints for the median of permutations problem, accepted to Journal of Discrete Applied Mathematics.

[18] A. Ali, M. Meilă, Experiments with kemeny ranking: What works when?, Mathematical Social Science 64 (2012) 28–40. doi:10.1016/j.mathsocsci.2011.08.008.

[19] F. Schalekamp, A. van Zuylen, Rank aggregation: Together we're strong, in: Proceedings of the Meeting on Algorithm Engineering & Experiments, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2009, pp. 38–51. doi:10.1137/1.9781611972894.4.

[20] N. Mattei, T. Walsh, Preflib: A library of preference data, Proceedings of 3rd International Conference on Algorithmic Decision Theory (ADT 2013), LNCS 8176 (2013) 259–270doi:10.1007/978-3-642-41575-3_20.

[21] R. Fisher, F. Yates, Statistical tables for biological, agricultural and medical research, 3rd edition, Oliver and Boyd, 1948, pp. 26–27.

[22] C. Mallows, Non-null ranking models. i, Biometrika 44 (1-2) (1957) 114–130. doi:10.1093/biomet/44.1-2.114.

[23] M. A. Fligner, J. S. Verducci, Distance based ranking models, Journal of the Royal Statistical Society. Series B (Methodological) (1986) 359–369.

[24] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, Optimization by simulated annealing, Science 220 (4598) (1983) 671–680. doi:10.1126/science.220.4598.671.

[25] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, N. Marshall, A. H. Teller, E. Telleri, Equation of state calculations by fast computing machines, Journal of Chemical Physics 21 (6) (1953) 1087–1092. doi:10.1063/1.1699114.

[26] Borda, Mémoire sur les élections au scrutin, Histoire de l'Académie Royale des Sciences.

[27] A. H. Copeland, A reasonable social welfare function, University of Michigan Seminar on Applications of Mathematics to the social sciences.

[28] P. Badal, A. Das, Efficient algorithms using subiterative convergence for kemeny ranking problem, Computers & Operations Research 98 (2018) 198–210.

[29] R. Milosz, S. Hamel, Medians of permutations: Building constraints, in: S. Govindarajan, A. Maheshwari (Eds.), Algorithms and Discrete Applied Mathematics, Vol. 9602 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2016, pp. 264–276. doi:10.1007/978-3-319-29221-2_23.

[30] R. Karp, Reducibility among combinatorial problems, in: R. Miller, J. Thatcher, J. Bohlinger (Eds.), Complexity of Computer Computations, The IBM Research Symposia Series, Springer, Boston, MA, 1972, pp. 85–103. doi:10.1007/978-1-4684-2001-2_9.

[31] M. Truchon, An extension of the condorcet criterion and kemeny orders, Internal Report, Université Laval.

[32] M. Truchon, Aggregation of rankings in figure skating, CIRPEE Working Paper.

[33] E. Emond, D. Mason, A new rank correlation coefficient with application to the consensus ranking problem, Journal of Multi-Criteria Decision Analysis 11 (1) (2002) 17–28. doi:10.1002/mcda.313.

[34] M. Meilă, K. Phadnis, A. Patterson, J. Bilmes, Consensus ranking under the exponential model, in: Proceedings of the 23rd Annual Conference on Uncertainty in Artificial Intelligence, 2007, pp. 285–294.