Université de Montréal

Étude combinatoire et algorithmique de médianes de permutations sous la distance de Kendall-Tau

par
Robin Milosz

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Rapport pour la partie orale
de l'examen pré-doctoral

août, 2017

TABLE DES MATIÈRES

LISTE DES FIGURES

NOTATION

π, σ	Permutations
n	Taille des permutations
S_n	Espace des permutations
$a \prec b$	Élément a préféré à l'élément b
$d_{KT}(\pi, \sigma)$	Distance de Kendall-τ
A	Ensemble de permutations de départ
m	Nombre de permutations de départ, taille de A
π^*	Permutation médiane
$M(A)$	Ensemble des permutations médianes
L	Matrice gauche
R	Matrice droite
R	Classement
$\mathscr{R}$	Ensemble de classements de départ
$Rank_n$	Espace des classements
$a = b$	Élément a à égalité avec l'élément b
E	Matrice égalité (pour le chapitre 4.5)
$\mathbb{S}^2$	Sphère

INTRODUCTION

Présentation du problème

Le problème de la médiane d'un ensemble A de m permutations des éléments $\{1,2,\ldots n\}$ sous la distance de Kendall-τ [26] [39] est un problème d'optimisation combinatoire où l'on veut trouver la permutation appelé médiane qui minimise la somme des distances de celle-ci aux m permutations de l'ensemble de départ A. La distance de Kendall-τ compte le nombre de paires d'éléments dont l'ordre relatif est en désaccord entre 2 permutations.

En science politique, ce problème est aussi connu sous le nom de "Kemeny Score Problem" [25] ou "méthode de Kemeny-Young". Dans ce contexte, on appelle consensus une permutations médiane. Le problème est ici presenté comme un système électoral où m électeurs vont individuellement ordonner une liste de n candidats selon leurs préférences.

Le problème est trivial à résoudre pour $m = 1$ et $m = 2$ et a été démontré NP-Difficile quand $m \geq 4$ avec m pair (en premier dans [20], puis corrigé dans [8]). La complexité demeure inconnue pour $m \geq 3$, m impair.

Applications

Le problème de la médiane de permutations trouve des applications en études politiques et dans plusieurs sciences.

Comme mentionné plus tôt, en politique, la médiane de permutations est un système électoral qui permet d'ordonner une liste de candidats (par opposition à un système à un seul gagnant) et qui possède plusieurs qualitées désirables dont le critère de la majorité, le critère de Condorcet et le critère de monotonicité.

En bio-informatique [36], la médiane de permutations sert dans le classement de protéines/gènes liées à une maladie [14] et la construction de cartes génétiques [19][5].

En science sociales, il sert dans l'ordonnancement de sujet à traiter ou de choix sociaux, ainsi que dans les classements sportifs [7].

En informatique, il peut servir à la meta-recherche sur le web et, plus anciennement, a servit à la détection de spam [20]. Il peut aussi servir dans la recommandation de choix[38] pour les utilisateur d'un service tel un fournisseur de films et émissions.

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.

Plan du rapport

Le rapport se divise en 5 chapitres qui sont insérés dans trois volets : A) la définition et le contexte du problème, B) le travail qui a déjà été complété dans le cadre du doctorat et C) les résultats préliminaires la planifications des futurs travaux. Le rapport est encadré par une introduction et une conclusion.

A)

Le chapitre 1 introduit le problème et son contexte. On y introduit aussi les définitions nécessaires à la compréhension du rapport.

Le chapitre 2 fait le tour des travaux qui ont été réalisé sur ce problème dans une revue de littérature.

B)

Le chapitre 3 est l'article qui a été présenté à la conférence IWOCA2017 (Internationnal Workshop On Combinatorial Algorithms), en juillet 2017.

C)

Le chapitre 4 fait un tour rapide des résultats préliminaires et travaux futurs considérś. En particulier :

La section 4.1 introduit la formulation du problème en programmation en nombres entiers et présente les premiers résultats de l'utilisation du solveur commercial ILOG CPLEX d'IBM.

La section 4.2 présente des idées intéressantes à investiguer dont une formule originale d'approximation en temps linéaire et ses applications potentielles

La section 4.3 présente une heuristique utilisant la nature du problème.

La section 4.4 présente des idées pour travailler avec des instances de grande taille.

La section 4.5 offre une généralisation du problème aux ensembles de permutations avec égalités.

Finalement, le chapitre 5 présente la planification du temps et des objectifs de recherche pour compléter le doctorat sur support d'un calendrier.

CHAPITRE 1

CONTEXTE ET DÉFINITIONS

1.1 Contexte

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 cherche à garder l'ordre relatif des paires au plus possible similaire à celui des votes, on peut observer un exemple à la figure 1.1.

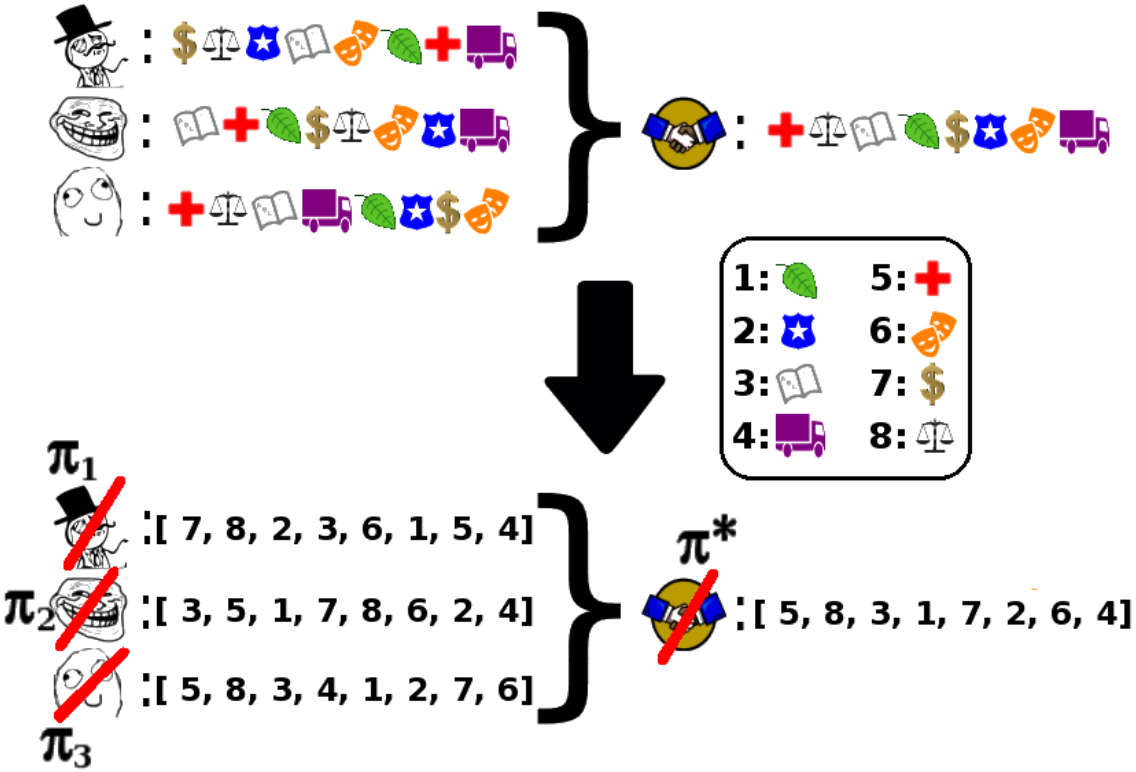

Figure 1.1 – Exemple d'introduction : trois citoyens ordonnent les sujets politiques selon leurs importances personnelles pour un éventuel débat ou pour une prise de décision. Sachant que les opinions sont différentes, on utilise le consensus de Kemeny pour trouver l'ordre consensus qui respectera le plus possible les ordres individuels. Dans cet exemple, il conviendra de traiter ces sujets dans l'ordre : "santé, justice, éducation, environnement, économie, sécurité, culture et transport". Figure tirée de mon mémoire "Étude de la médiane de permutations sous la distance de Kendall-Tau".

Le problème a été prouvé NP-Difficile dans [20]. 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. Nous allons commencer ici en formalisant le problème dans ce qui suit.

1.2 Définitions

Une **permutation** π de n éléments est une bijection de l'ensemble $\{1,2,\ldots,n\}$ sur lui-même, c'est donc un ordre total des éléments $\{1,2,\ldots,n\}$. Elle peut aussi correspondre à une liste ordonnée d'objets. On dénote une permutation π de $\{1,2,\ldots,n\}$ par $\pi = \pi_1\pi_2\ldots\pi_n$ où $\pi[i]$ dénote la position de l'entier i dans π et où π_j dénote l'entier à la position j. (voir Figure 1.2).

$$\pi = [\ 3, 5, 1, 7, 8, 6, 2, 4]$$

Figure 1.2 – π est une permutation de $n = 8$. Dans π, l'élément 5 se retrouve avant l'élément 1. On note l'ordre entre 5 et 1 : $5 \prec 1$ car 5 précède 1. On note $\pi[7] = 4$ pour dire que l'élément 7 se trouve à la position 4 et on note $\pi_4 = 7$ pour dire qu'à la position 4 se trouve l'élément 7. Figure tirée de mon mémoire "Étude de la médiane de permutations sous la distance de Kendall-Tau".

On définit l'**ordre entre deux éléments** (ou l'ordre relatif dans une paire d'éléments) d'une permutation comme suit : $a \prec b$ signifie que a est préféré à b (ou que a est à gauche de b, ou que a est avant b) alors que $b \prec a$ signifie que b est préféré à a. Comme une permutation est un ordre total, chaque paire d'éléments a un ordre bien défini.

L'**ensemble des permutations** de départ A est l'ensemble des permutations qui caractérise une instance du problème. Ce sont les votes sur les candidats. Il y a $\#A = m$ permutations dans l'ensemble A et elles sont toutes de taille identique n. Ces deux premières caractéristiques vont déjà être utiles à discriminer la difficuté du problème. On peut permettre la répétition de permutations dans l'ensemble et dans ce cas A devient un multi-ensemble ou une collection de permutations.

Pour caractériser un ensemble (ou une collection) de permutations on définit deux matrices : L la **matrice "gauche"** et R 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. Formellement, $L[i][j](A)$ (resp. $R[i][j](A)$) dénote le nombre de permutations de l'ensemble où l'élément i est à gauche de l'élément j, $i < j$ (resp. i à droite de j, $j < i$). Par définition on a que : $L[i][j](A) + R[i][j](A) = m = |A|$ et $L[i][j](A) = R[j][i](A)$. La Figure 1.3 donne les matrices gauches et droites de l'exemple de la Figure 1.1.

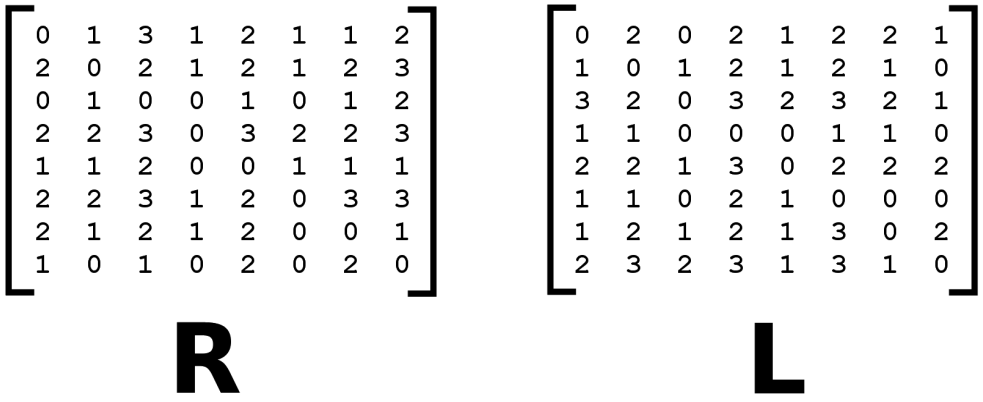

Figure 1.3 – Matrices Droite R et Gauche L pour l'ensemble A de la Figure 1.1 d'introduction.

L'**espace des permutations** S_n est l'ensemble contenant toutes les permutations de taille n (voir Figure 1.4 pour un exemple visuel). C'est notre espace de recherche.

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

$$d_{KT}(\pi,\sigma) = \#\{(i,j)|i < j \wedge (\pi[i] < \pi[j] \wedge \sigma[i] > \sigma[j]) \vee (\pi[i] > \pi[j] \wedge \sigma[i] < \sigma[j]).\}$$

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. la permutation π lue de droite à gauche. 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'algorithmne de tri 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 nécessaires pour passer d'une permutation à une autre (voir figure 1.5).

On généralise la distance de Kendall-τ entre 2 permutations à une distance entre une

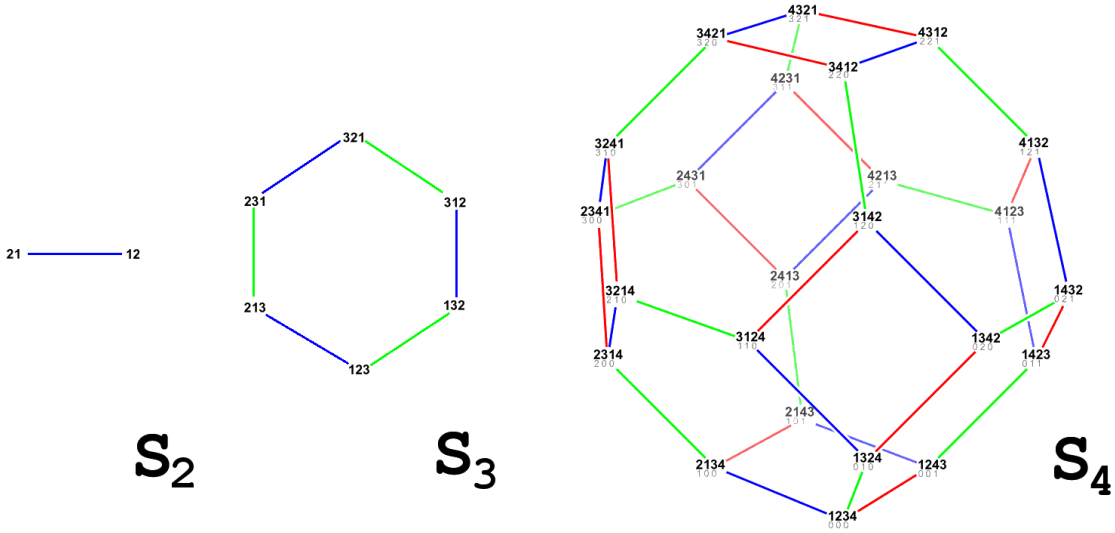

Figure 1.4 – L'espace des permutations, exemples pour S_2, S_3 et S_4. Ici, une arête est créé entre deux permutations si elles ne diffèrent que d'une transposition d'éléments adjacents. Ces polytopes ainsi créés sont appelés des permutahedrons. La couleur des arêtes indique la position des transpositions : 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.

permutation et un ensemble de permutations de la façon suivante : Soit une permutation $\pi \in S_n$ et un ensemble de permutations $A \subset S_n$ alors :

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

Le **problème de la médiane de permutations** vise à trouver la ou les permutations de S_n qui minimisent la distance entre elle et l'ensemble A. Formellement : Soit $A \subset S_n$ un ensemble de permutations, trouver l'ensemble $M(A)$ des permutations $\pi^* \in S_n$ tel que $d_{KT}(\pi^*, A) \leq d_{KT}(\pi, A), \forall \pi \in S_n$. Dans une forme équivalente retrouvée dans la littérature, on cherche une permutation médiane π^* :

$$\pi^* = argmin_{\pi \in S_n}\{d_{KT}(\pi, A)\}.$$

Une **permutation médiane** π^* d'un ensemble A est une permutation qui est la plus proche de A sous la distance de Kentall-τ. La **distance médiane** $dM_{K\tau}$ est la distance entre π^* et A, c'est souvent le "score" qu'on tente de minimiser dans les heuristiques.

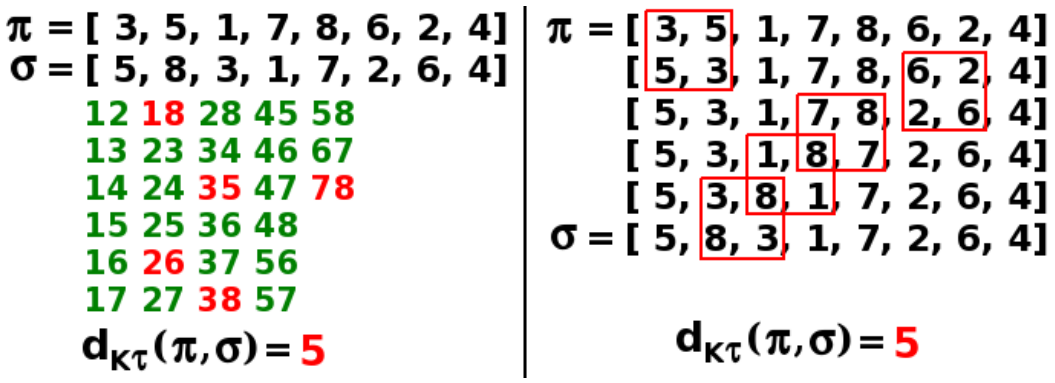

Figure 1.5 – Distance de Kentall-τ entre deux permutations π et σ. À gauche, les paires en rouges représentent les désaccords et leur nombre est la distance. C'est équivalent au nombre de transpositions d'éléments adjacents nécessaires pour passer d'une permutation à l'autre. À droite on peut observer les transpositions d'éléments adjacents encadrés en rouge. Figure tirée de mon mémoire "Étude de la médiane de permutations sous la distance de Kendall-Tau".

$$dM_{K\tau} = min_{\pi \in S_n}\{d_{KT}(\pi, A)\}$$

L'ensemble **M(A)** est l'ensemble de toutes les permutations médianes de A. C'est un ensemble dont la taille varie énormément d'une instance à une autre.

Finalement, 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 3.1) ou dans une autre variante, trouver seulement la distance médiane.

CHAPITRE 2

REVUE DE LITTÉRATURE

La distance de Kendall-τ a été introduite pour la première fois en 1938 par Maurice Kendall[26], un statisticien britannique.

Le problème de la médiane d'un ensemble de m permutations $A \subset S_n$ sous la distance de Kendall-τ a été introduit par John George Kemeny [25], un mathématicien et informaticien en 1959 avec l'objectif de créer une méthode pour trouver un consensus entre divers ordres de préférence.

Le problème de la médiane de permutations sous la distance de Kendall-τ est aussi retrouvé dans la littérature sous les noms de : "méthode de Kemeny-Young" ("Kemeny-Young method")[42], "règle de Kemeny" ("Kemeny rule")[18], "règle de classement de Kemeny" ("Kemeny ranking rule")[3],"problème de classement de Kemeny" ("Kemeny ranking problem")[21][3], "le problème du score de Kemeny" ("Kemeny Score Problem")[21][7], "le problème d'agrégation de classements de Kemeny" ("Kemeny Rank Aggregation problem") [7], "le problème d'agrégation de classements" ("Rank Aggregation problem") [35] et "classement par popularité VoteFair" ("VoteFair popularity ranking") http://www.votefair.org/.

En science politique, le problème est vue comme un système électoral où m électeurs vont individuellement ordonner une liste de n candidats selon leurs préférences.

C'est sous cette version du problème que Peyton Young a réalisé les premiers travaux sur le problème en 1978 avec A. Levenglick ; dans [42], il montre que ce système électoral est le seul qui respecte le critère "de renforcement" (si les votes sont séparés en plusieurs groupes et qu'un même candidat est gagnant dans chaque groupe alors ce candidat sera gagnant dans l'éléction) et le critère de Condorcet [15] (voir Section 2.5) et il poursuit 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[43][44].

Après ces travaux pionniers sur le problème, quelques années sont passées et le pro-

blème est revenu en force à l'attention de la communauté depuis les 15 dernières années.

Ces travaux on trait principalement à l'approximation d'une médiane, aux algorithmes à paramètre fixe, à la résolution exacte, à l'étude de la complexité du probèleme, des bornes inférieures et de la réduction d'espace de recherche.

Dans ce qui suit, nous ferons un bref résumé des travaux antérieurs réalisés dans chacune de ces directions de recherche.

2.1 Complexité du problème

La compléxité du problème de la médiane de permutations a été démontrée NP-Difficile en premier dans[6] puis dans [20] pour $m \geq 4$ et m pair. La preuve de [20] se base sur la réduction du "Feedback edge set problem"[23] au problème de la médiane de permutations. Le "Feedback edge set problem" consiste à déterminer s'il existe un sous-ensemble d'arêtes de taille inférieur ou égal à L qui, lorsque retiré du graphe, rend celui-ci acyclique. Dans la preuve, quatre permutations sont construites à partir du graphe du premier problème et une solution au problème de la médiane sur ces quatres permutations peut être convertie en solution pour le "Feedback edge set problem", et inversement.

Une petite erreur dans cette preuve a été par la suite corrigé dans [8]. La compléxité pour le cas $m \geq 3$, m impair reste encore inconnue.

2.2 Méthodes heuristiques

Dans [20], 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 [34][11]. Chaque candidat est un état. D'un état quelconque l'heuristique peut transiter vers n'importe quel autre état avec probabilté 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 (repartir d'un autre état) 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 probabilté stationnaire. L'heuristique est calculable en temps polynomial et est une 2-approximation c'est-à-dire que la distance de Kendall-τ de la solution trouvée à l'ensemble de départ est au plus 2 fois plus grande que la distance d'une vraie médiane à l'ensemble.

Les auteurs de [35] utilisent la même idée et proposent **MC4Approx**, une version approximée de MC4 qui est calculé plus rapidement en évitant de calculer la distribution stationnaire.

Les auteurs de [35][1] ont aussi démontré que 2 heuristiques très simples sont aussi des 2-approximations. La première heuristique **pick-a-perm**[35] consiste à choisir comme médiane (approximée) d'un ensemble de permutations $A \subset S_n$ une permutation aléatoirement dans A alors que sa version déterministe, **BestOfA**[35], consiste à choisir comme médiane (approximée) la permutation de A qui minimise la distance à A.

D'autres auteurs ont étudié une approche gloutonne pour la résolution du problème. Donc étant donné une permutation choisie, ces approches vont localement optimiser la distance de Kendall-τ à l'ensemble de permutations $A \subset S_n$ dans le but de se rapprocher de la médiane.

L'heuristique **2-opt** (voir Section 4.4) consiste à appliquer sur une permutation choisie, des échanges d'éléments adjacents jusqu'au moment où aucun tel échange ne fasse descendre la distance à l'ensemble A. De la même façon, **k-opt** est la généralisation où k éléments consécutifs sont réarrangés pour faire descendre la distance à A.

L'heuristique de **Chanas** [13] consiste à appliquer 2-opt une première fois sur une permutation choisie, inverser la permutation résultat puis réappliquer 2-opt.

Plusieurs algorithmes de tri ont été testé comme heuristique sur le problème de la médiane. Ces algorithmes utilisent l'idée d'*ordre majoritaire* entre paires d'éléments $(i, j),\ 1 \leq i \neq j \leq n$ qui est l'ordre relatif $i \prec j$ ou $j \prec i$ qui apparaît dans la majorité des permutations de l'ensemble de départ A. Si A contient un nombre impair de permutations, il y a toujours un ordre majoritaire, sinon, il est possible que $i \prec j$ dans la moitié des permutations de A et $j \prec i$ dans le reste. À ce moment là, il n'y a pas d'ordre majoritaire pour la paire d'éléments (i, j).

QuickSort[1][2][35] 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 gauche ou à droite selon l'ordre majoritaire par rapport au pivot pivot. Les éléments avec qui n'ont pas d'ordre majoritaire par rapport au pivot sont classés arbitrairement. La méthode est répétée récurssivement sur les éléments à gauche du pivot et ceux à droite. Une version déterministe[40] 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 gauches et ceux à droite . Une version plus rapide consiste à évaluer $log(n)$ éléments comme pivots et semble aussi efficace[35]. La version probabiliste est d'un facteur d'approximation $11/7$ et celle déterministe est d'un facteur d'approximation $8/5$.

MergeSort[35], basé sur le tri fusion, "tri" les éléments en les scindant en deux liste à trier. Une fusion de deux liste 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[35][20], 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 heuristique à base de tri ont l'inconvénient de trier "localement" les éléments. Cet aspect est valide quand les nombres sont tous ordonnés dans un ordre total mais ne fonctionne plus avec les éléments des permutations de l'ensemble A avec lesquels la transitivité n'est plus valide ($i \prec j, j \prec k \not\Rightarrow i \prec k$ dans A). D'autres méthodes heuristiques ont été développés qui trient les éléments mais selon d'autres critères (et avec une approche plus globale) et sont présenté dans ce qui suit.

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.

La méthode **Borda** ou "BordaCount" [10] consiste à ordonner les éléments selon leur posistion moyenne dans les permutations de l'ensemble de départ A.

La méthode **Copeland** (Copeland 1951)[17] consiste à ordonner les éléments selon leur nombre de victoires 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 A.

Finalement, des auteurs ont exploré des idées d'heuristiques en limitant les ressources des algorithmes de résolution exacte de type Branch-and-Bound.

Dans [28], Meila et al. dérivent une heuristique de recherche en faisseaux (Beam Search) en limitant le nombre de branchement à chaque étape de leur algorithme Branch-and-Bound. Dans cette heuristique, les noeuds sont des preéfixes de permutations et le branchement consiste à ajouter le prochain élément au préfixe.

Dans [18] est proposé une heuristique gloutonne basée sur le graphe orienté de tournoi correspondant à l'ensemble A. En premier lieu, le critère de Condorcet étendu (voir la section 2.5) 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. À chaque iteration, une fermeture transitive est appliquée sur l'ensemble des paires.

Comme vous venez de lire, 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.

2.3 Schéma d'approximation en temps polynomial

Un schéma d'approximation en temps polynomial (PTAS) est donné dans [27]. C'est un algorithme qui donne une approximation de facteur $(1+\varepsilon)OPT$ où OPT est la distance médiane cherchée dans un temps qui dépends de ε. Pour une instance du probleme,

i.e. un ensemble de permutations $A \subset S_n$, le temps d'exécution est de l'ordre de :

$$O(n^3 logn(log(1/b)+1/\varepsilon))+n2^{(1/(\varepsilon b)^6)}$$

pour une version non-déterministe, b étant une constante fixée par l'algorithme. Le temps est exponnentiel en $1/\varepsilon$, 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 et réduit considérablement le facteur d'approximation pour le problème (qui était constant avec les heuristiques menionnées ci-haut) mais est d'intérêt théorique seulement([40], [37], [31] et [2]).

2.4 Algorithmes à paramètres fixes

Plusieurs travaux ont suggéré un lien entre des paramètres de similarité et la diffculté de résolution du problème pour des ensembles $A \subset S_n$ de m permutations de $\{1,...,n\}$. La compléxité paramétrée a alors été étudiée dans une série de plusieurs articles récents.

Premièrement, dans [31] Betzler et al. proposent un algorithme qui dépend de la distance médiane (ou le score de Kemeny) k et dont la complexité est dans l'ordre de $O(1.53^k + n^2 m)$. L'algorithme est basé sur la kernelization et un arbre de recherche avec profondeur limitée. Un deuxième algorithme de programmation dynamique est paramétré par la distance maximale d entre deux permutation de l'ensemble de départ A et se calcule en $O((3d+1)! \times dlogd \times nm)$.

Dans [32], 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 A et se calcule en $O(32^{r_{max}} \times (r_{max}^2 m + r_{max} m^2))$.

Dans [37], 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 compte le nombre de médianes en $(36^{\frac{k}{m}})$.

Dans [24], Karpinski et Schudy ont obtenu un algorithme en $O(2^{O(\sqrt{\frac{k}{m}})} + n^{O(1)})$.

Dans [33], un algorithme en $O(4^{\frac{k}{m}})$ est prosposé qui permet l'énumération des permutations médianes.

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.

2.5 Réduction de l'espace de recherche pour la médiane

Dans [7], Betzler et al. proposent une réduction d'espace basé sur des éléments-pivots qui permettent de scinder le problème en deux sous-problèmes. Dans leur travail, une paire de éléments $(i, j),\ 1 \leq i \neq j \leq n$ est dite "non-dirty" si le nombre de permutations de l'ensemble de départ $A \subset S_n$ appuyant un des deux ordres ($i \prec j$ ou $j \prec i$) dépasse une certaine proportion. Entre autres, il investiguent le seuil de 0.75, et donc si dans plus des $\frac{3}{4}$ des permutations de l'ensemble de départ 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 préféré à lui dans la majorité des permutations de A vont être placé avant lui et les autres vont être placé après lui dans une permutation médiane. Cette méthode sera appelée dans ce qui suit la **3/4-Majority Rule**.

Dans[9], 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 départ. Dans le second théorème, ils prouvent qu'une paire d'éléments qui se trouvent toujours dans le même ordre dans les permutations de départ conservera cet ordre dans une permutation médiane. Ce deuxième théorème sera appelé dans ce qui suit le **Always Theorem**.

Une extension du critère de Condorcet est présenté dans [39]. Le **critère de Condorcet??** dit que si un élément est préféré à majorité à chacun des autres éléments alors il doit être

le 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** 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.

Dans [29], travail réalisé pendant ma maîtrise, est proposée une réduction d'espace qui réponds 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.

Elle se base sur le fait que si deux éléments ont un ordre majortaire dans les permutations de A et qu'ils sont proches (en terme de posistions 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 A dans lesquelles ils sont dans l'ordre minoritaire, alors cette paires c'éléments va conserver son ordre majoritaire dans les permutations médianes. Cette méthode sera référée comme le **Major Order Theorem**.

Dans une version journal en révision de l'article, on implémente une version calculable très rapidement grace à une implémentation vectorielle. 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 constraintes ont été apportée au lecteur. Une grande difficulté est rencontré sur les ensembles de permutations aléatoires qui offrent peu de similitude entre elles. Pour autant, ces méthodes sembles faire partie de la solution gagnante (en combinant aec 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 exacte de type Branch-and-Bound.

2.6 Borne inférieure sur la distance médiane

Dans [18], Davenport et Kalagnanam ont proposé une première borne inférieure intuitive pour la distance de Kendall-τ d'une médiane π^* à un ensemble de permutations $A \subset S_n$. Elle consiste à comptabliser, 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 en désaccord avec toutes les paires d'éléments dans les permutations dans lesquelles elles sont dans un ordre minoritaire) :

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

où $R_{ij}(A)$ est la matrice de droite définit à la Section 1.

Une seconde borne inférieure est obtenue en augmentant la première borne de la manière qui suit. On considère le graphe orienté avec poids sur les arcs $G = (V,E)$, 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 $w(e_{ij}) = R_{ji}(A)$ et $w(e_{ji}) = R_{ij}(A)$. Ce graphe est une représentation de l'ensemble des permutations de 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 feedback arc set problem [23]).

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 A : $w(e'_{ij}) = w(e_{ij}) - min\{R_{ij}(A), R_{ji}(A)\}$, $\forall 1 \leq i < j \leq n$. Cela ce fait parallèlement avec le calcul de la premiere borne inférieure $LowerBound_0$, qui compte ces annulations de poids d'arcs opposés.

Soit $DC_{G'}$ un 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 $[n]$ dans une permutation médiane, il y aura

au moins un arc par cycle qui ne pourra pas etre 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 d'utiliser que des cycles disjoints de longueur 3.

Dans [16], Conitzer, Davenport et Kalagnanam reprennent ces travaux et poussent plus loin la borne inférieure en démontrant que les cycles ne doivent pas être disjoints nécessairement.

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 être aussi utilisés pour des raisons de vitesse de calcul.

Finalement, Conitzer et al. proposent une borne inférieure basée sur la programmation linéaire simplement en enlevant la contrainte d'intégralité de la modélisation en programmation en nombre entiers du problème (voir section 4.1) et testent cette borne avec CLPEX. Le résultat est concluant : c'est la borne inférieure la plus stricte.

2.7 Approches exacte pour résoudre le problème

Deux approches exactes pour résoudre le problème ont été étudiées soient une approche Branch-and-Bound et une formulation du problème en programmation linéaire en nombres entiers.

Un algorithme du Branch-and-Bound pour le problème est introduit dans [18]. Dans cet algorithme, une recherche en profondeur est effectuée et chaque noeud de recherche

est un ensemble de paires ordonnées. À l'étape du branchement, une paire d'éléments dont l'ordre n'a pas encore été déterminé est choisie puis ordonnée. Les paires (i, j) dont la différence entre $R_{ij}(A)$ et $R_{ji}(A)$ est la plus élevée sont choisie en priorité. Quand la paire est ordonnée, une étape de fermeture transitive est appliquée. À l'étape d'élagage, la borne inférieure est évaluée sur la nouvelle solution, et si elle dépasse le score de la meilleure permutation trouvée jusqu'à présent, elle est éliminée. Des tests sont effectués sur des instances allant jusqu'à $n = 15$ éléments, avec m permutations dans l'ensemble de départ, m impair et avec une certaine simlarité entre les permutations de l'ensemble de départ A. Les auteurs dérivent une heuristique gloutonne à partir de l'algorithme en ne permettant pas le retour en arrière.

Dans [28] un second algorithme Branch-and-Bound est proposé dans lequel les noeuds de l'arbre de recherche du Branch-and-Bound 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 ameliorée de [16]. Le prochain noeud a être séléctionné est celui qui la borne inférieure la plus petite (méthode meilleur d'abord). Des test sont effectués sur des instances aléatoires de taille allant jusqu'à $n = 10$. Les auteurs dérivent une heuristique à partir de l'algorithme en limitant le nombre de noeuds par niveau de l'arbre de recherche ("Beam Search").

La formulation en programmation en nombres entiers (IP ou ILP) pour le problème de la médiane a été introduite dans [16] où ILOG CPLEX est utilisé pour résoudre le la formulation IP. Des tests sont effectués sur des instances allant jusqu'à $n = 40$ éléments, mais toujours avec $m = 5$ permutations. Dans ce travail, les auteurs prouvent une meilleure heuristique d'élagage pour le Branch-and-Bound en améliorant la borne inférieure.

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

Dans [22], travail réalisé pendant ma maîtrise, nous avons investigué deux classes d'instances du problème dites auto-médianes ($A = M(A)$). Les instances de la première

classe sont caracterisées par le fait que l'ensemble des permutations A est construit à partir d'une permutation sur laquelle sont effectué des mouvements circulaires pour obtenir toutes les autres permutations de A. La seconde classe comprends les instances dans lesquelles A est l'ensemble S_n. Des opérations de combinaison de ces classes sont présentées et conservent la propriété d'auto-médiane.

Si l'approche Branch-and-Bound a été étudiée par plusieurs auteurs, l'approche par programmation en nombres entiers semble sous-estimée et pourrait bénificier de plus d'attention étant donné sa performance observée dans la prochaine section. Ceci dit, la prochaine section va aborder la comparaison entre ces algorithmes de résolution et diverses heuristiques.

2.8 Travaux de comparaison

Une premiere étude comparative a été faite en 2009 par Schalekamp et Zuylen dans [35]. Beaucoup d'heuristiques et divers combinaisons de celles-ci y sont évaluées sur des données provenant de recherches sur le web. L'étude recommande l'utilisation des heursitiques Borda, Copeland, QuickSort et MC4Approx (décrites à la Section 2.2) pour obtenir un approximation rapidement (recommandations basées sur le temps de calcul et la qualité de l'approximation). Les auteurs montrent aussi que la solution obtenue en combinant chacune de ces quatres heuristiques et l'heuristique InsertionSort donne un résultat encore meilleur i.e. une solution plus proche de la vrai médiane.

Dans [3] (2012), 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 A : h_{borda}/h_{low} qui est le rapport entre l'approximation de la distance médiane donnée par l'heuristique Borda et la borne inférieure triviale (voir $LowerBound_0$ plus haut). Pour résoudre le problème, le solveur ILOG CPLEX d'IBM ressort gagnant de leurs expé-

riences. Borda et Copeland, deux heuristiques simples et rapides, sont conseillées pour plusieurs cas et des suggestions de solutions de rechange pour le solveur CPLEX, sont présentées : un algorithme de recherche locale et leur solveur Branch-and-Bound puis des combinaisons de ceux-ci. Le travail recommande d'investiguer la réduction d'espace (en prétraitement) en combinaison avec les algorithmes et heuristiques discutées.

2.9 Contributions futures

Nos contributions futures vont s'orienter principalement sur les heuristiques, la réduction d'espace de recherche et les algorithmes exactes. Le prochain chapitre présente en détails la première contribution publié de cette thèse [30].

CHAPITRE 3

TRAVAUX PUBLIÉS

3.1 Heuristique, Branch-and-bound et réduction d'espace de recherche

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 simmulé ("Simulated Annealing" ou "SA") qui a été bien paramétrisé à l'aide de nombreux tests. L'heuristique donne une bonne approximation qui est souvent la solution optimale même (plus de 96% du temps) pour les tailles d'instances étudiées.

La seconde approche combine la réduction d'espace de recherche par les Major Order Theorems[29] avec la borne inférieure de Conitzer et al.[16] pour donner une borne inférieure plus stricte. Lorsque combiné à l'approximation par recuit simmulé, 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 précédantes pour guider l'exploration et couper les sous-espace de recherche qui ne contiennent pas de solution optimales.

L'article "Heuristic, Branch-and-Bound Solver and Improved Space Reduction for the Median of Permutations Problem" a été accepté et présenté à la conférence International Workshop On Combinatorial Algorithms, July 2017, Newcastle, New South Wales, Australia (IWOCA2017) et se trouve dans les proceedings de celle-ci [30]. Une version journal de cet article, contenant la combinaison de des deux premières approches avec une formulation en nombres entiers du problème et CPLEX est en cours de rédaction.

L'article et son annexe se trouve en annexe de ce rapport.

CHAPITRE 4

RÉSULTATS PRÉLIMINAIRES ET TRAVAUX FUTURS

4.1 Programmation en nombres entiers

4.1.1 Motivation

Dans l'objectif de résoudre le problème de la médiane exactement, deux approches se retrouvent dans la littérature : celle d'un algorithme Branch-and-Bound et celle de la résolution de la formulation en programmation en nombres entiers (IP). Si la première approche a reçu plus d'attention dans les travaux ([18] [28] [22]), ce n'est pas le cas de la deuxième approche qui n'est pas allée plus loin que la formulation et sa résolution simple([16] [28]). En effet, les solveurs d'IP CPLEX et Gurobi sont appelé sur des instances IP du problème de manière générique, sans aucun aspect différent d'un problème d'ordonnancement linéaire (LOP). Sachant que ces même travaux ([16] [28]) classent la deuxième approche comme la meilleure manière de résoudre le problème, il serait alors intéressant d'étudier davantage cette approche.

4.1.2 Modélisation

On peut modéliser le problème de la médiane d'un ensemble de permutations $A \subset S_n$ en programmation en nombres entiers (Integer Programming - IP) en utilisant la modélisation standard du problème d'ordonnancement linéaire (LOP) :

$$minimiser : \sum_{i<j,\ i,j\in\{1,\ldots,n\}} R_{ij}(A)x_{ij} \tag{4.1}$$

sujet aux contraintes :

$$x_{ij} + x_{ji} = 1, \quad i \neq j,\ i,j \in \{1,\ldots,n\} \tag{4.2}$$

$$x_{ij} + x_{jk} + x_{ki} \geq 1, \quad i \neq j \neq k,\ i,j,k \in \{1,\ldots,n\} \tag{4.3}$$

$$x_{ij} \in [0,1] \quad i \neq j,\ i,j \in \{1,...,n\} \tag{4.4}$$

$$x_{ij}\ entier \quad i \neq j,\ i,j \in \{1,...,n\} \tag{4.5}$$

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 fonction de coût (4.1), $R_{ij}(A)$ indique le nombre de permutations dans A dans lesquelles i est à droite de j.

La contrainte (4.2) $x_{ij} + x_{ji} = 1$ force à avoir un ordre soit $i \prec j$ soit $j \prec i$ ou entre les deux. La contrainte (4.3) $x_{ij} + x_{jk} + x_{ki} \geq 1$ force la transitivité de la relation de précédence : si $i \prec j$ et $j \prec k$ alors $i \prec k$. La contrainte (4.4) $x_{ij} \in [0,1]$ oblige les variables d'ordre à être bornées. Puis la contrainte d'intégralité (4.5) force à avoir un ordre total en (4.2) : si les x_{ij} sont entiers alors on a soit $i \prec j$ ou $j \prec i$.

La résolution de ce problème IP donne un score qui est celui de la distance médiane et des assignations de variables que l'on peut convertir en une permutation médiane pour l'ensemble A.

Si on enlève la dernière contrainte (contrainte d'intégralité), on obtient une borne inférieure intéressante pour le problème par la méthode du simplexe.

4.1.3 Expérimentations avec CPLEX

Une implémentation du solveur CPLEX ILOG d'IBM dans Java (Concert Technology) a été faite pour résoudre le problème de la médiane de permutations dans cette modélisation "MIP" - Mixed Integer Programming. Cette implémentation est le meilleur solveur pour ce problème jusqu'à présent selon nous et selon [3]. La borne inférieure calculée avec la méthode du simplexe est aussi la meilleur borne d'après nos observations pratiques et celles de [16].

CPLEX permet l'introduction d'une solution de départ (MIP Start - MST) et il s'avère que l'heuristique du recuit simulé (SA) paramétrée introduite dans la Section 3 permet d'accélérer les calculs de CPLEX en lui donnant une solution approximative.

Les contraintes développées dans le même chapitre peuvent être intégrées directement dans la modélisation IP sous forme $x_{ij} = 1$ si $i \prec j$ est une contrainte trouvée, ce qui rend le problème plus facile à résoudre.

L'introduction de la solution de départ de SA et des constraintes n'est pas automatiquement liée à une augmentation de la performance dans tous les cas comme on peut voir a la Figure 4.1. Par contre, en moyenne, il est avantageux de fournir ces connaissances supplémentaires à CPLEX : le temps moyen de résolution est le plus bas avec la solution de départ et les contraintes.

Notez que le temps d'exécution de l'heuristique SA est négligeable en pratique par rapport au temps d'exécution de CPLEX. Le temps demandé pour calculer les contraintes est plus grand sur les petites instances mais la tendance s'inverse complètement pour les tailles plus grandes ($n \geq 80$) au point où il ne représente qu'une petite fraction du temps d'exécution de CPLEX.

taille du problème:		caractéristiques:			
m	n	original	mst	constraints	constraints+mst
3	60	515.56	646.21	449.62	579.77
4	60	454.89	578.87	203.38	332.12
5	60	14288.28	8788.22	9688.74	6183.98
3	70	1563.79	1788.11	1396.94	1606.77
4	70	720.38	909.04	116.75	332.31
5	70	1931.09	2134.26	2063.31	2270.31
3	80	304903.18	143389.36	106089.00	77493.30
4	80	1470.65	1771.20	1209.36	1517.77
5	80	33023.36	4540.35	26113.12	4975.42
3	90	4140.55	4586.23	3475.34	3907.65
4	90	197793.46	15106.97	132881.12	115424.61
5	90	2781696.72	2383961.55	679621.16	908664.74
3	100	775017.20	104621.75	160247.34	37931.90
4	100	4182.20	4807.44	5100.42	5630.65
5	100	136065.61	14990.22	101878.41	15293.40
somme des temps:		4257766.92	2692619.78	1230534.01	1182144.70

Figure 4.1 – Temps de résolution en ticks (unité de temps déterministe de CPLEX) de CPLEX pour des instances du problème de $m = 3,4,5$ permutations et de taille allant de $n = 60$ à $n = 100$ éléments. Pour chaque problème, sont résolues : la modélisation originale, celle avec une solution de départ fournie par l'heuristique SA (mst), celle avec des contraintes ajoutées (constraints) puis celle intégrant les deux (constraints+mst). En vert sont indiqués les meilleurs options pour résoudre chaque instance. La ligne du bas fait la somme des temps pour résoudre toutes les instances avec chaque option. On remarque que le fait d'ajouter une solution de départ et des contraintes au modèle permet d'obtenir la meilleure performance en temps. Exécution de CPLEX sur 1 processus (1 thread).

Cela démontre qu'une combinaison de méthodes (IP, SA et contraintes) semble être

la façon optimale de résoudre le problème de la médiane de permutations.

Ce travail préliminaire sera inclu dans l'article journal donnant suite à l'article de conférence[30].

Plusieurs directions s'ouvrent pour des travaux futurs, entre autres étudier la formulation même du problème pourrait avoir une grande répercussion sur la résolution. De plus, l'étude des paramètres optimaux de CPLEX serait d'un bon intérêt pour celui qui veut résoudre régulièrement plusieurs problèmes de la médiane de permutations. Finalement, les méthodes de coupes (Gomory, Chvátal), la décomposition de Benders, la relaxation Lagrangienne et la génération de colonnes sont des directions qui restent a être exploré de cette approche en programmation en nombres entiers.

4.2 Formule d'approximation

4.2.1 Motivation

Comme discuté précédement, la distance de Kendall-τ entre deux permutations de taille n se calcule naïvement en $O(n^2)$ en comparant toutes les paires et en $O(nlogn)$ en utilisant une version modifiée de l'algorithmne de tri fusion. Nous nous sommes posés la question à savoir s'il était possible d'avoir une bonne approximation de cette distance en temps linéaire.

Une façon envisagée est de considérer les permutations de taille n comme des coordonnées de $\mathbb{R}^n$ et de mesurer la distance euclidienne. L'idée vient du fait que l'espace des permutations (le permutahedron) est fermé sur lui-même comme observé sur la Figure 4.2. Par analogie au cercle dans $\mathbb{R}^2$, le rapport de la distance sur la surface d'un cercle (arc de cercle) et de la distance euclidienne des coordonnées (corde de cercle) entre deux points peut être vu comme une fonction, tel qu'observé sur la Figure 4.3. Nous allons ainsi approximer la distance de Kendall-τ en utilisant une distance euclidienne calculée à partir des permutations (voir Figure 4.4).

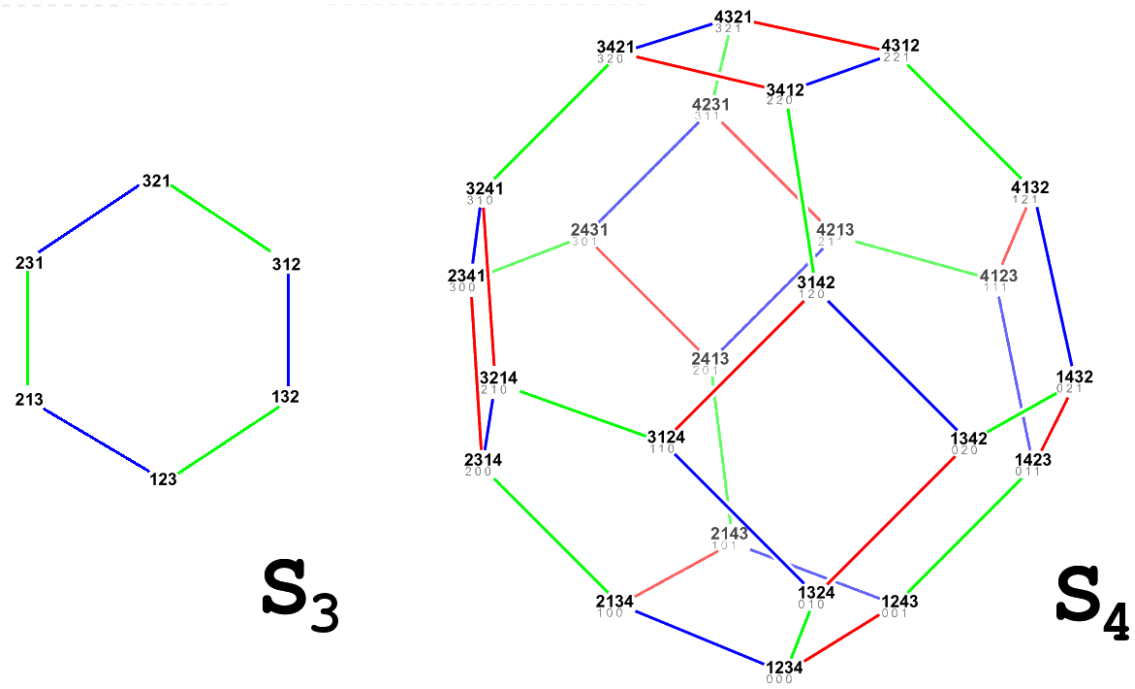

Figure 4.2 – Permutahedrons pour $n = 3$ et $n = 4$. Une arête bleue signifie une transposition des 2 premiers éléments, arête verte : des 2 éléments suivants et arête rouge : des 2 derniers éléments.

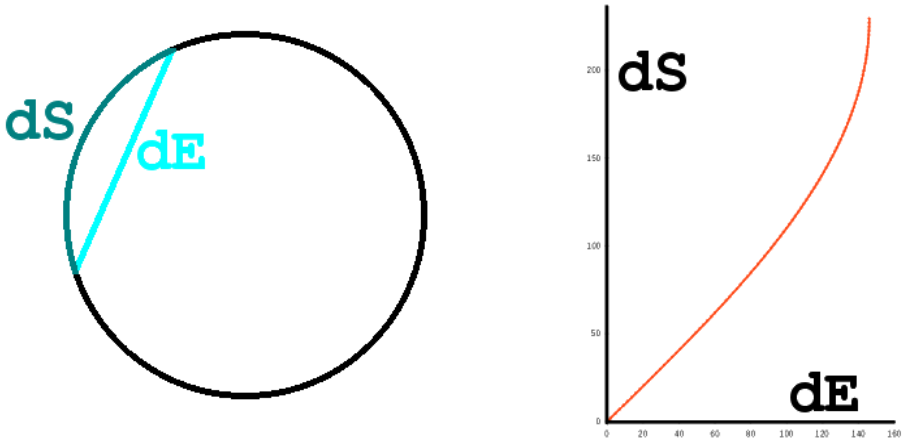

Figure 4.3 – Distance sur la surface du cercle dS par rapport à la distance euclidienne des coordonnées dE dans la relaxation dans $\mathbb{R}^2$.

4.2.2 Démarche et résulat

Dans ce qui suit, la démarche consistera à approximer la distance de Kendall-τ par l'arc de cercle qui relie les deux coordonnées. Après avoir établi la fonction qui donne

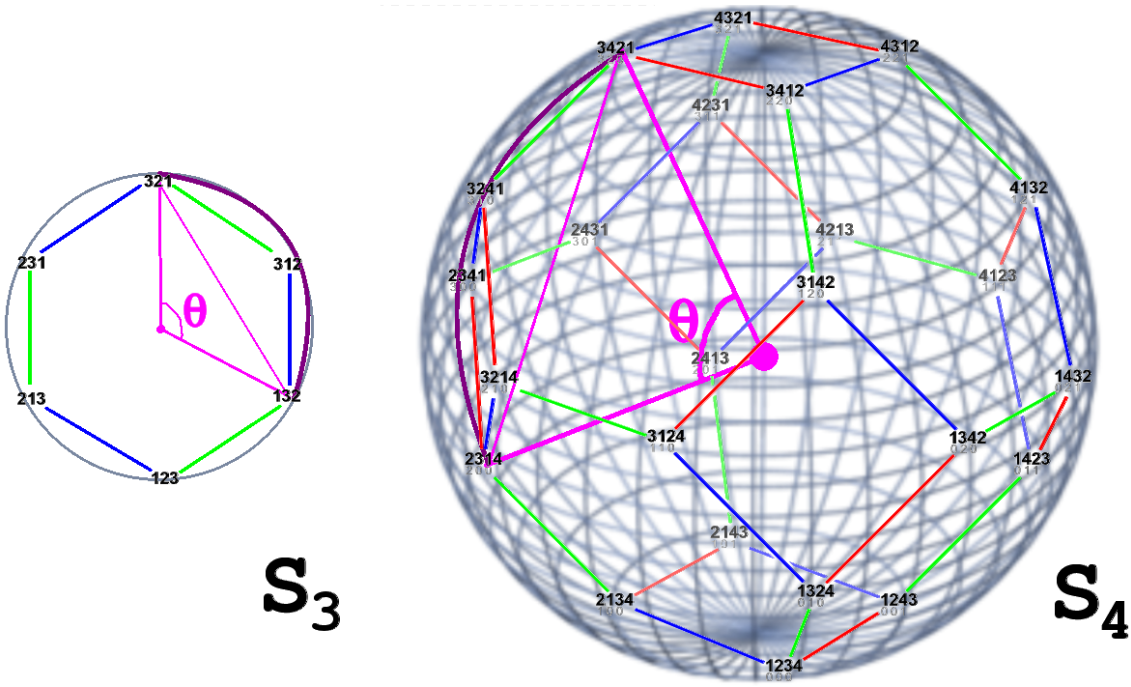

Figure 4.4 – Permutahedrons pour $n = 3$ et $n = 4$ et sphères superposées. La distance de Kendall-τ entre deux permutations se calcule en comptant le nombre minimal d'arêtes pour passer d'une permutation à une autre. La distance euclidienne se mesure entre les coordonnées des deux permutations. L'angle θ est mesuré à partir du centre de la sphère qui contient les coordonnées de toutes les permutations de S_n.

les coordonnées des permutations, il faudra calculer la distance euclidienne, en déduire l'arc de cercle correspondant grâce à l'angle qui le forme et obtenir l'approximation de la distance de Kendall-τ en multipliant par le rapport des distances maximales sur le permutahedron et sur la sphère :

En premier lieu, il faut établir une fonction injective entre les permutations et les coordonnées telle que chaque permutation est envoyé sur son unique coordonnée $f : S_n \to \mathbb{R}^n$ et telle que deux permutations à distance de Kendall-τ de 1 vont être voisine dans l'espace des coordonnées. Une solution est de simplement d'utiliser la permutation inverse (inverse par composition) d'une permutation pour obtenir ses coordonnées [45]. La composition de permutations est définie comme suit : soit π et σ deux permutations, la composition $\sigma \times \pi$ envoie l'élément i sur l'élément $\sigma(\pi(i))$ où $\pi(i) = \pi_i$. La permutation inverse π^{-1} est alors l'unique permutation telle que $\pi^{-1} \times \pi = I$ où I est la

permutation identité. Il est intéressant de remarquer que la permutation inverse s'obtient facilement : $\pi_i^{-1} = \pi[i]$ où $\pi[i]$ dénote la position de i dans π et où π_i dénote l'élément de π à la position i. Dans cette relation, deux permutations distantes de 1 (distance de Kendall-τ) vont être distantes de $\sqrt{2}$ dans l'espace des coordonnées euclidiennes.

Par exemple, pour $\pi = (2,4,1,3)$ et $\sigma = (2,1,4,3)$ on a les inverses $\pi^{-1} = (3,1,4,2)$ et $\sigma^{-1} = (2,1,4,3)$ donc les coordonnées associées aux permutations sont $f(\pi) = [3,1,4,2]$ et $f(\sigma) = [2,1,4,3]$. La distance euclidienne entre les deux coordonnées est $\sqrt{(3-2)^2 + (1-1)^2 + (4-4)^2 + (2-3)^2} = \sqrt{1+0+0+1} = \sqrt{2}$. On va donc définir la distance euclidienne entre 2 permutations comme ci :

$$d_{eucl}(\pi, \sigma) = \sqrt{\sum_{i \in \{1,\ldots,n\}} \left(\pi_i^{-1} - \sigma_i^{-1}\right)^2}.$$

En second lieu, il faut noter que la distance maximale de Kendall-τ entre 2 permutations est de $\frac{n(n-1)}{2}$ et se produit lorsque les 2 permutations sont à l'opposé sur le permutahedron. Par exemple, $\pi = \{1,2,3,4\}$ et $\sigma = \{4,3,2,1\}$. Il faut noter aussi que la distance maximale euclidienne entre des coordonnées de permutations se produit dans le même cas où les coordonnées sont opposées. Avec un peu de calcul et en observant le triangle de Pascal, on arrive à une distance euclidienne maximale de $\sqrt{2\binom{n+1}{3}}$.

En troisième lieu, on se rappele la formule de trigonométrie :

$$c^2 = a^2 + b^2 - 2ab\cos\theta$$

On peut isoler l'angle, ce qui nous donne :

$$\theta = \arccos\left(\frac{a^2 + b^2 - c^2}{2ab}\right)$$

Dans notre cas, les côtés du triangle sont les rayons du cercle et la distance euclidienne mesurée entre les deux coordonnées, comme montré sur la figure 4.5. On peut

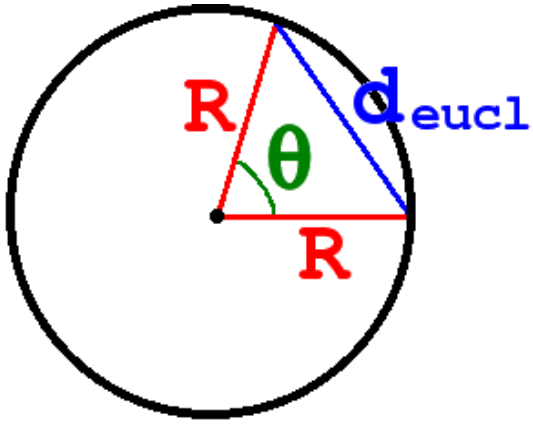

Figure 4.5 – Relation entre l'angle θ et le côté opposé dans un triangle. Le triangle est formé par deux points sur la surface du cercle et le point au centre du cercle.

donc simplifier la formule ($a = b = R$ et $c = d_{eucl}$) :

$$\theta = \arccos\left(\frac{2R^2 - d_{eucl}^2}{2R^2}\right)$$

$$\theta = \arccos\left(1 - \frac{d_{eucl}^2}{2R^2}\right)$$

Le diamètre du cercle utilisé ici est la distance euclidienne maximale, alors le rayon est $R = \sqrt{2\binom{n+1}{3}}/2$. On a donc :

$$\theta = \arccos\left(1 - \frac{d_{eucl}^2}{2\left(\frac{\sqrt{2\binom{n+1}{3}}}{2}\right)^2}\right)$$

$$\theta = \arccos\left(1 - \frac{d_{eucl}^2}{2\left(\frac{2\binom{n+1}{3}}{4}\right)}\right)$$

$$\theta = \arccos\left(1 - \frac{d_{euc}^2}{\binom{n+1}{3}}\right)$$

On peut maintenant approximer la distance de Kendall-τ entre 2 permutations en multipliant la distance sur la surface du cercle $d_S = \theta R$ par le rapport de la distance maximale de Kendall-τ ($n(n-1)/2$) divisée par la distance maximale sur la surface du

cercle ($R\pi$ où $R = \sqrt{2\binom{n+1}{3}}/2$) :

$$d_{K\tau}(\pi,\sigma) \simeq d_S \times \frac{\text{distance maximale de Kendall-}\tau}{\text{distance maximale sur le cercle}}$$

$$d_{K\tau}(\pi,\sigma) \simeq d_S \times \frac{n(n-1)/2}{R\pi}$$

$$d_{K\tau}(\pi,\sigma) \simeq \theta R \times \frac{n(n-1)}{2R\pi}$$

$$d_{K\tau}(\pi,\sigma) \simeq \theta \times \frac{n(n-1)}{2\pi}$$

$$d_{K\tau}(\pi,\sigma) \simeq \arccos\left(1 - \frac{d^2_{eucl}}{\binom{n+1}{3}}\right) \times \frac{n(n-1)}{2\pi}.$$

La formule d'approximation est au final :

$$d_{K\tau}(\pi,\sigma) \simeq \arccos\left(1 - \frac{\sum_{i\in\{1,\dots,n\}}\left(\pi_i^{-1} - \sigma_i^{-1}\right)^2}{\binom{n+1}{3}}\right) \times \frac{n(n-1)}{2\pi}.$$

Les figures suivantes 4.6 4.7 donnent une idée qualitative de la précision de la formule d'approximation. Les figures ont été construites en générant des paires de permutations : pour chaque paire, la distance de Kendall-τ et la distance euclidienne ont été calculées. Elles sont indiquées en bleu. La formule d'approximation a été ensuite tracée en orange.

L'intérêt de cette formule d'approximation ne réside pas seulement dans le temps linéaire nécessaire pour le calcul mais aussi dans la continuité : on passe d'une fonction discrète à une fonction continue. Les avantages de cette relaxation continue sont de manipuler des objets qui ne sont plus des permutations mais des coordonnées et de pouvoir utiliser une heuristique de type hill-climbing comme décrite dans la prochaine section.

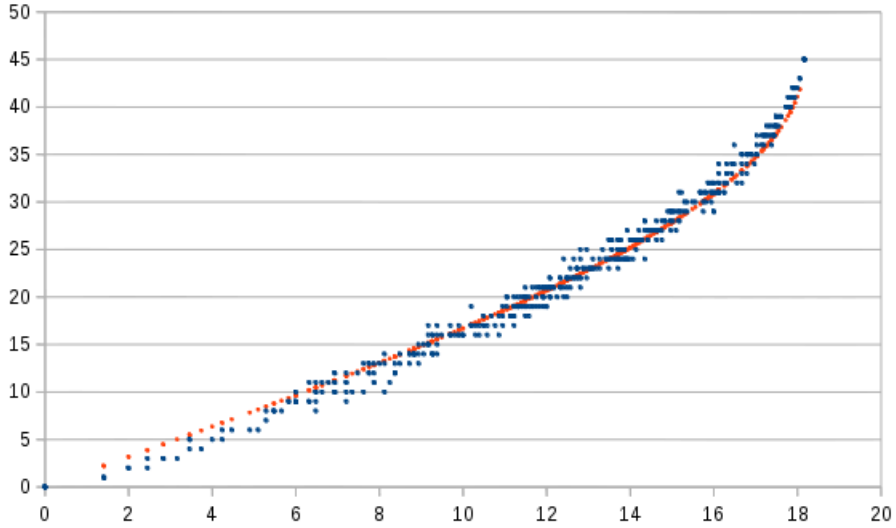

Figure 4.6 – Distance de Kendall-τ en fonction de la distance euclidienne pour des permutations de $n = 10$. En bleu, sont indiquées les vraies distance de Kendall-τ par rapport au distances euclidiennes mesurées alors que la formule d'approximation est indiquée en orange. Le taux d'écart de la formule d'approximation par rapport à la vraie distance est de 6.00% en moyenne, statistique générée sur 500 paires de permutations aléatoires.

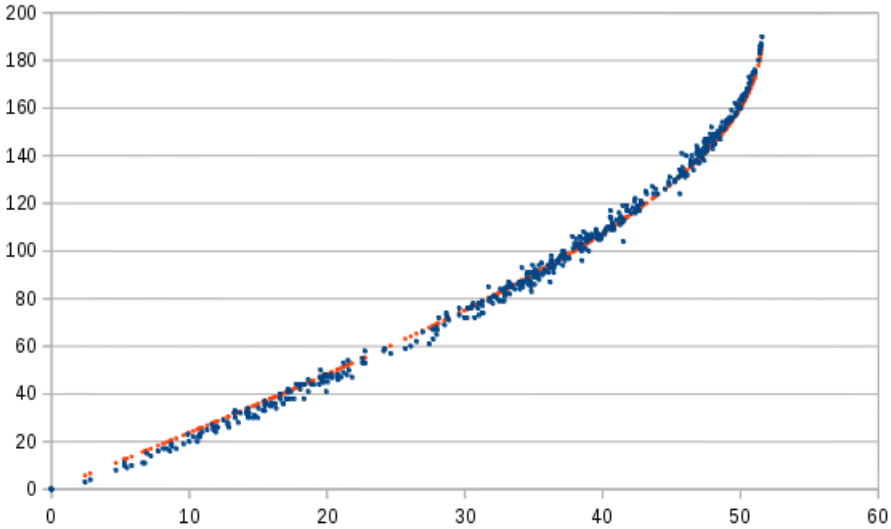

Figure 4.7 – Distance de Kendall-τ en fonction de la distance euclidienne pour des permutations de $n = 20$. En bleu, sont indiquées les vraies distance de Kendall-τ par rapport au distances euclidiennes mesurées alors que la formule d'approximation est indiquée en orange. Le taux d'écart de la formule d'approximation par rapport à la vraie distance est de 3.93% en moyenne, statistique générée sur 500 paires de permutations aléatoires.

4.2.3 Heuristique de descente de gradient

Une heuristique basée sur la formule d'approximation est en cours de développement. Le principe est de faire une descente de gradient dans l'espace des coordonnées en minimisant la somme des distances de Kendall-τ approximées et de projeter la solution sur le permutahedron.

La formule d'approximation est dérivable et la dérivée donne un vecteur dans lequel on effectue un pas pour minimiser la somme des distances. Cette étape est répétée itérativement et le pas est réduit en suivant une échelle logarithmique (1, 0.5, 0.25, ...) à chaque k itérations, $k \leq 10$. La solution converge vers un minimum global. L'heuristique va ensuite retourner la permutation dont la coordonnée euclidienne est la plus proche de la coordonnée de la solution. Une analogie avec le cercle dans $\mathbb{R}^2$ est présentée à la figure 4.8.

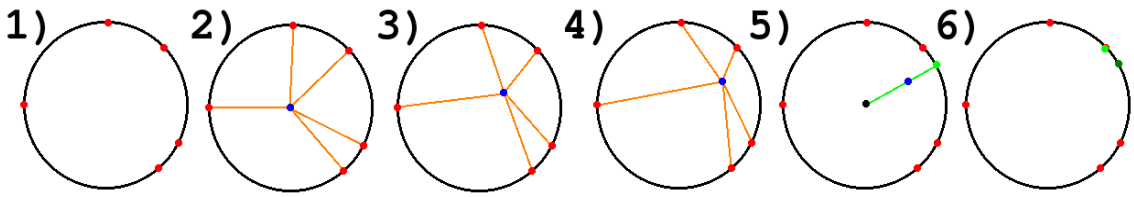

Figure 4.8 – Analogie au problème de la médiane de points sur un cercle. 1) Un problème est donné dans lequel il faut trouver le point sur le cercle qui minimise la somme des distances par arc de cercle. Dans 2), 3) et 4) on fait une descente de gradient sur la fonction qui somme les distances approximées. Ces distances approximées sont calculées à partir des distances euclidiennes mesurées en utilisant une fonction telle que présentée dans la figure 4.3. En 5), on projette la solution trouvée sur la surface du cercle, on prenant le point du cercle qui est le plus proche de la solution. En 6), on raffine la solution.

Cette heuristique a l'avantage de diriger la solution vers la direction générale du minimum global car la monotonicité des fonctions de distance et la relaxation continue enlèvent la difficulté des minimums locaux. Malgré l'imprécision de la formule d'approximation, la solution approximative obtenue de l'heuristique peut servir de point de départ pour une seconde heuristique qui serait basée sur la recherche locale ou le raffinement de solution. L'intérêt est dans le temps linéaire nécessaire au calcul de la distance (approximée) de Kendall-τ et le nombre d'itérations en $logn$ dans la descente de gradient, ce qui est avantageux pour résoudre des instances plus grandes.

Des tests préliminaires nous ont donné des résultats comparables aux heuristiques Copeland et Borda de la Section 2.2.

Nous travaillons présentement à raffiner cette heuristique pour en améliorer l'efficacité. La voie étudié est l'ajout d'un élément correctif calculable en temps linéaire (par exemple la distance de manhattan, les profils des différences de positions des éléments, l'écart-type des différences de positions) à la formule d'approximation dans le but d'en améliorer la précision,

4.3 Heuristique "Median Game"

4.3.1 Motivation

La majorité des heuristiques qu'on utilise ne sont pas spécifiques au problème de la médiane de permutations mais peuvent être utilisées aussi pour le problème du Linear Ordering Problem (LOP) qui est une des généralisations du problème de la médiane qui consiste à trouver l'ordonnencement des éléments qui minimise la somme des coûts attribués à l'ordre relatif pour chaque paire d'éléments (c_{ij} si $i \prec j$ et c_{ji} si $j \prec i$). L'idée ici était de développer une heuristique qui tient de la nature du problème de la médiane où l'on veut minimiser la somme des distances à un ensemble.

Le principe est d'exécuter des mouvements sur une permutation "solution" en direction de permutations aléatoirement tirées de l'ensemble départ. Par analogie, si on voudrait approximer la médiane d'une suite s de nombres, on pourrait commencer avec un nombre p, puis de piger itérativement un nombre q de la suite s et exécuter un pas dans sa direction. Par exemple, si $q < p$ alors $p \leftarrow p - \alpha$ et si $q > p$ alors $p \leftarrow p + \alpha$, pour un petit α.

Le nom "Mediane Game" a été choisi du fait qu'on peut imaginer, dans un contexte de vote en science politique, un jeu où chaque voteur va à tour de rôle rapprocher la solution du débat d'un petit pas vers son intérêt/choix personnel en argumentant. Dans ce jeu, on effectue plusieurs tour de table, dans lesquels chacun à son tour de parole. On veut qu'au final, la solution converge vers un consensus (ou se rapproche de celui-ci).

4.3.2 Démarche et résultats

L'heuristique commence avec une permutation aléatoire (la solution de départ) puis, itérativement fait des tours de rapprochement jusqu'à un nombre maximal d'itérations déterminé au départ. Un tour de rapprochement consiste à effectuer une transposition d'éléments adjacents qui diminue la distance de Kendall-τ entre la solution courante et une permutation de l'ensemble de départ A et ce pour chaque permutation de A, donc m transpositions par tour (qui peuvent s'annuler). L'ordre des permutations de A est aléatoire à chaque tour et la position de la transposition est aussi aléatoire pour éviter de faire du sur-place. Cette version non-déterministe semble performer un peu mieux dans nos tests préliminaires qu'une la version déterministe où l'ordre des permutations est préétablis.

Comme dans la section précédente, cette heuristique se trouve aussi entre les méthodes de Copeland et de BordaCount, en terme de performances, d'après des premières observations.

Les figures 4.9 4.10 4.11 donnent un support visuel du comportement de l'heuristique.

Il serait intéressant d'investiguer des différents types de mouvements de rapprochement avec cette méthode du tour de rapprochement pour obtenir une meilleure heuristique. Il faudrait aussi dérterminer le nombre d'itérations à faire pour arriver à un résultat intéressant.

4.4 Grandes instances

4.4.1 Contexte et motivation

Le terme "grandes instances" n'est pas bien défini mais on peut imaginer des instances de plusieurs permutations de tailles $n > 100$ éléments, voir de centaines d'éléments ou même de milliers d'éléments. Plus concrètement, l'intérêt serait dans l'étude des instances qui ne sont plus résolvable en temps pratique. De telles instances apparaissent lorsque le nombre de candidats à trier est très grand comme en bio-informatique

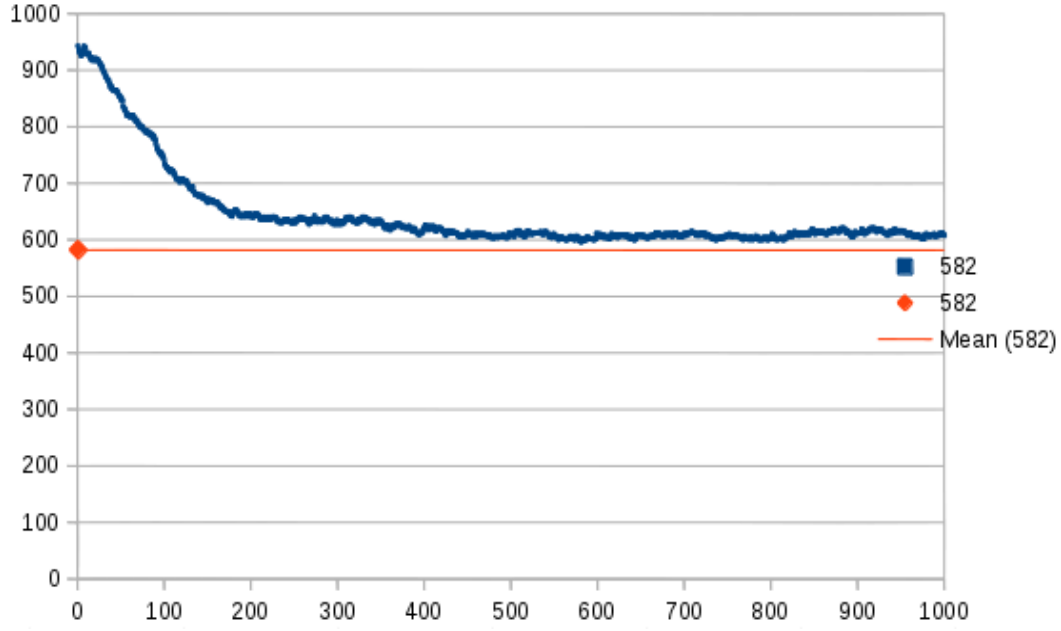

Figure 4.9 – Score de la solution courante au courant des itérations de l'heuristique Median Game. En orange est indiqué la distance médiane (le score optimal = 582). L'ensemble de départ est constitué de $m = 4$ permutations de $n = 30$ éléments. Le minimum enregistré par l'heuristique est de 596 (soit à 2.41% de l'optimal).

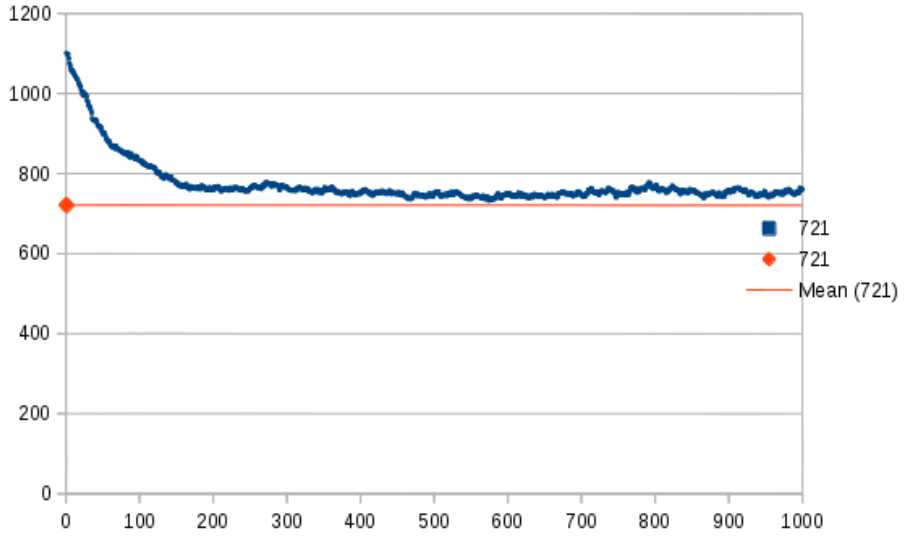

Figure 4.10 – Score de la solution courante au courant des itérations de l'heuristique Median Game. En orange est indiqué la distance médiane (le score optimal = 721). L'ensemble de départ est constitué de $m = 5$ permutations de $n = 30$ éléments. Le minimum enregistré par l'heuristique est de 733 (soit à 1.66% de l'optimal).

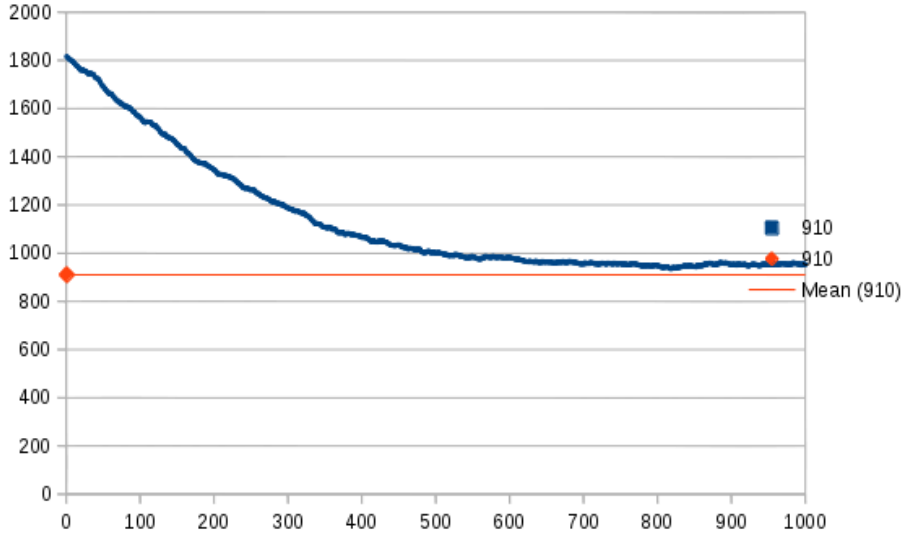

Figure 4.11 – Score de la solution courante au courant des itérations de l'heuristique Median Game. En orange est indiqué la distance médiane (le score optimal = 910). L'ensemble de départ est constitué de $m = 3$ permutations de $n = 50$ éléments. Le minimum enregistré par l'heuristique est de 936 (soit à 2.86% de l'optimal).

(classements de gènes liés à une maladie).

Il faut noter que c'est le nombre d'éléments dans les permutations et non le nombre de permutations dans l'ensemble de départ A qui influence le plus la difficulté du problème.

4.4.2 Pistes à explorer

4.4.2.1 Heuristiques

Une première approche serait d'étudier l'utilisation d'heuristiques sur les grandes instances, entre autres les heuristiques Borda et Copeland décrites dans la Section 2.2, l'heuristique du recuit simulé de la Section 3.1, l'heuristique MedianGame de la Section 4.3 et l'heuristique de la descente de gradient de la formule d'approximation de la Section 4.2, puis des combinaisons de celles-ci.

4.4.2.2 Window-optimizer

Une autre approche serait de faire de l'optimisation dans une fenêtre sur une permutation. Cela consisterait à verifier si un réarrangement des éléments dans la fenêtre considérée peut réduire la distance de cette permutation aux permutations de départ. La fenêtre serait déplacée plusieurs fois sur la permutation. On utiliserait les outils disponibles qui peuvent résoudre ces petites instances rapidement. La permutation initiale pourrait provenir de n'importe quelle heuristique. L'optimisation par fenêtre aurait une vision plus locale pour compléter.

Des premiers tests ont été réalisés sur une combinaison d'une heuristique BordaCount ou Copeland avec une heuristique de raffinement 2-opt ou 3-opt (voir Section 2.2) dans lesquelles une fenêtre de taille 2 ou 3 est repassé sur une permutation donnée par BordaCount ou Copeland jusqu'à ce qu'il n'y ai plus de changement pouvant faire diminuer la distance. Cette combinaison est plus performante que les heuristiques BordaCount ou Copeland utilisées seules. Les heuristiques 2-opt et 3-opt quant à elles, ne sont pas intéressantes lorsque utilisées seules.

Il serait intéressant d'étudier la propriété d'une permutation qui est k-opt (dont n'importe quels k éléments adjacents dans la permutation sont dans un ordre optimal - dont aucun réarrangement ne peut diminuer la distance). On pourrait se demander s'il y a une garantie de facteur d'approximation pour un certain ratio de k par rapport à n.

D'après des premières observations, les heuristiques 2-opt et 3-opt sont moins performantes que BestOfA sur des instances moyennes, BestOfA étant l'heuristique qui consiste à simplement retourner la permutations de A la plus proche de A (voir Section 2.2) et qui a été démontrée être d'un facteur d'approximation 2 (la distance de la solution trouvée est au plus 2 fois la distance médiane).

4.5 Classements avec égalité

4.5.1 Contexte

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 (exemple à la figure 4.12) et 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 $j \prec i$, $\forall i, j,\ i \neq j$).

Les classements ont aussi leur utilité dans le domaine de la bio-informatique[4] [14] où plusieurs méthodes donnent des classements différents de gènes liées à une certaine maladie (exemple à la Figure 4.13).

4.5.2 Définitions

Un classement est une permutation dans laquelle on introduit la relation d'égalité entre deux éléments. Plusieurs éléments peuvent partager la même position, ils sont alors dans le même "bucket" (panier) comme on peut le voir dans les classements de la Figure 4.12. Chaque paire d'éléments a un ordre de préférence ($i \prec j$, $j \prec j$ ou $i = j$) et l'ordonnancement est transitif :

- $i \prec j, j \prec k \rightarrow i \prec k$
- $i = j, j = k \rightarrow i = k$
- $i \prec j, j = k \rightarrow i \prec k$
- $i = j, j \prec k \rightarrow i \prec k$.

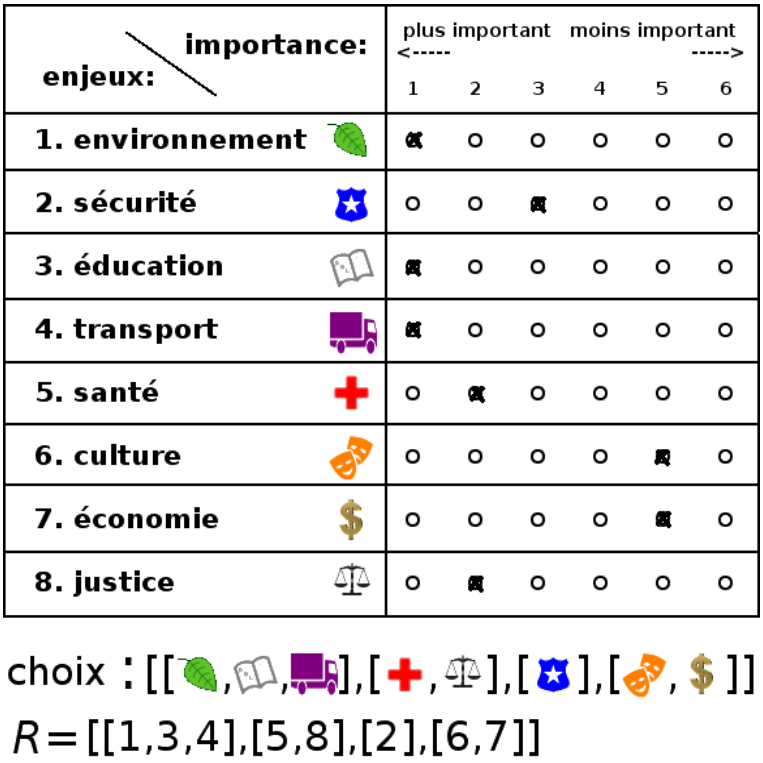

Figure 4.12 – Exemple de questionnaire qui engendre des classements pour des enjeux de société. Ce type de questionnaire est très populaire en sciences sociales et nous est assez familier. Un remplissage correct de ce questionnaire peut être interprété comme un classement avec égalités. Dans le choix illustré, l'environnement, l'éducation et le transport figurent tous en première place donc sont placés dans le premier "panier", suivi de la santé et la justice dans le deuxième panier, etc . Le classement R est obtenu en assignant une numérotation aux enjeux. Figure tirée de mon mémoire "Étude de la médiane de permutations sous la distance de Kendall-Tau".

L'ensemble de tous les classements de taille n est noté $Rank_n$ et est de taille :

$$|Rank_n| = \sum_{k=1}^{n} k! \left\{ {n \atop k} \right\} \approx \frac{n!}{2(\log 2)^{n+1}}$$

(i.e. le nombre de fonctions surjectives de $\{1,...,n\}$ à $\{1,...,k\}$) où $\left\{ {n \atop k} \right\}$ sont les nombres de Stirling de deuxième espèce). La formule d'approximation est due à [41] et la séquence des tailles $|Rank_n|$ se retrouve sur le site OEIS https://oeis.org/A000670 nommé "les nombres de Fubini".

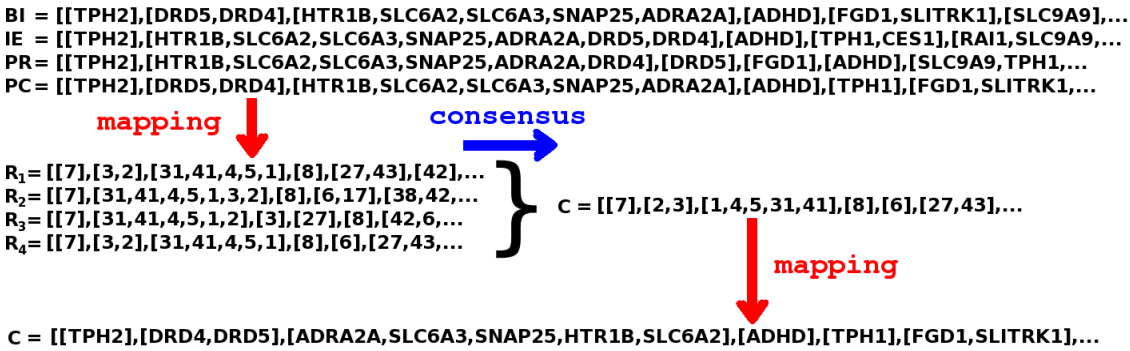

Figure 4.13 – Exemple de classements de gènes. Dans cette figure, quatre classements de gènes liés à la maladie de l'ADHD (Attention Deficit Hyperactivity Disorder) sont présentés dans des groupements ordonnés selon l'importance. On modélise le problème en associant un entier à chaque gène et on trouve le consensus de ces classements. On obtient alors un classement qui fait ressortir les gènes importants. On peut ensuite se baser sur ce classement pour orienter des futures recherches. Cet exemple provient de données provenant du site de BioConcert http://bioguide-project.net/bioconsert/. Méthodes de classement des gènes : BI=Bioggle, IE=In Edge, PR=Page Rank et PC=Path Count. Figure tirée de mon mémoire "Étude de la médiane de permutations sous la distance de Kendall-Tau".

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

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

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 $p \in [0,1]$ 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.

On défini la distance d'un classement R à un ensemble de classements $\mathscr{R}$ comme la somme des distances entre le classement R et chaque classement R_i de l'ensemble $\mathscr{R}$.

Formellement :

$$K(R,\mathscr{R}) = \sum_{R_i \in \mathscr{R}} K(R,R_i)$$

Le problème de la médiane de classements se définit mathématiquement comme suit : Soit un ensemble de classements $\mathscr{R} = \{R_1, \dots R_t\}$, $R_i \in Rank_n$, trouver l'ensemble des classements $R^* \in Rank_n$ tel que :

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

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

4.5.3 Stage Mitacs

De octobre 2017 à mars 2018, je serais en stage au Laboratoire de Recherche en Informatique (LRI) de l'Université Paris-Sud en France grâce à la bourse Mitacs. Les travaux envisagés ont trait aux classements et sur les instances de grandes tailles (Section 4.4).

Bryan Brancotte, un doctorant du LRI qui a fait un stage à Montréal en 2014, a étudié dans [12] des adaptations de nombreuses heuristiques pour le problème de la médiane de permutations au problème de la médiane de classements et a fait un important travail de comparaison entre ces heuristiques. 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. Les travaux qui seront réalisés dans le cadre du stage Mitacs s'inscrivent donc dans la suite des travaux de B.Brancotte.

Les idées à explorer ici sont de transposer les connaissances du cas simple des permutations aux classements dans le but d'élaborer des nouvelles méthodes et algorithmes :

- les heuristiques dont le recuit simulé[30] (Section 3.1), MedianGame (Section 4.3), BordaCount et Copeland (Section 2.2) et autres heuristiques
- les contraintes Always [9], 3/4-Majority Rule[7], Major Order Theorem (MOT)[29] (toutes de la Section 2.5) et Lower Upper Bound Constraints (LUBC)[30] (Section 3.1)

- les bornes inférieures Conitzer[16] et extensions[30] (Section 2.6)
- la programmation en nombre entiers et l'utilisation de CPLEX (Section 4.1)
- le calcul rapide ($O(nlogn)$) de la distance de Kendall-τ généralisée (Section 1)
- vérifier si une formule d'approximation est possible (Section 4.2)

Il serait aussi intéressant par la suite de tester nos méthodes et algorithmes sur les données réelles provenant du site de PrefLib http://www.preflib.org/. Ce site contient des ensembles de classements tirés de plusieurs sources et dont les tailles varient autant par le nombre d'éléments que par le nombre de permutations.

4.5.4 Autres généralisations

Si les classements avec égalité sont une généralisation des permutations, on pourrait vouloir aussi étudier les classements où tous les candidats ne sont pas nécessairement présents dans tous les classements. On peut s'imaginer plusieurs courses dans lesquelles participent des athlètes mais ces athlètes ne participent pas obligatoirement à toutes les courses. Présentement, les façons de gérer les candidats absents sont soit l'unification, qui consiste à ajouter les candidats absents à la fin de chaque classement (ce qui peut s'avérer injuste dans un contexte sportif) ou de simplement les ignorer et considérer les candidats présents dans tous les classements (la projection). Cette généralisation peut s'appliquer sur des classements avec égalité ou sur des permutations.

Une autre généralisation possible est d'attribuer une pénalité p différente de 1 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".

Une dernière généralisation possible est celle avec des ordre partiels. C'est-à-dire où l'ordre de préférence entre deux candidats peut être absent. Un ordre partiel est un relation entre des candidats qui respecte la transitivité. Donc à la place d'un ensemble de départ de permutations ou classements, on a un ensemble d'ordres partiels qu'on peut cumuler dans une matrice ou un graphe de tournoi.

CHAPITRE 5

CALENDRIER

session	période	objectifs
A-2017	septembre	investigation de la programmation en nombres entiers (IP), investigation de la formule d'approximation, début du travail sur les instances de grande taille, article journal des résultats de IWOCA avec la formulation IP et CPLEX
	octobre	(Stage en France) transposition des connaissances du cas des permutations au cas des classements avec égalités, prise de connaissance des travaux réalisé dans le laboratoire d'accueil (LRI), travail sur les instances de grande taille
	novembre-déc.	(Stage en France) travail sur les classements avec égalités, applications au big data en bio-informatique, travail sur les instances de grande taille
H-2018	janvier-fevrier	(Stage en France) rédaction d'un article sur les classements avec égalités et sur le travail réalise pendant le stage
	mars	rédaction du rapport sur le stage en France pour Mitacs
	avril	début de la planification et l'organisation de la thèse
E-2018	début	étude du problème avec la perspective de la programmation nombres entiers (Integer Programming), début rédaction de la thèse
	milieu	IP, rédaction de la thèse
	fin	rédaction d'un article sur les travaux réalisés en hiver et été, rédaction de la thèse
A-2018	début	finalisation de la thèse et depôt
	milieu	corrections et préparation de la soutenance
	fin	soutenance

BIBLIOGRAPHIE

[1] Charikar M. Newman A. Ailon, N. Aggregating inconsistent information : ranking and clustering. *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, STOC. Association for Computing Machinery (ACM)*, 6, 2005.

[2] N. Ailon, M. Charikar et N. Newman. Aggregating inconsistent information : ranking and clustering. *Journal of the ACM*, 55(5):pp 1–27, 2008.

[3] A. Ali et M. Meila. Experiments with kemeny ranking : What works when ? *Mathematical Social Sciences*, 64, 2012.

[4] A. Denise B. Brancotte, B. Rance et S. Cohen-Boulakia. Conqur-bio : Consensus rankings with query reformulation for biological data. *Data Integration in the Life Sciences*, LNCS 8574, 2014.

[5] P. S. Schnable B. N. Jackson et S. Aluru. Consensus genetic maps as median orders from inconsistent sources. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 5(2), 2008.

[6] Tovey C.A. Trick M.A. Bartholdi III, J. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare*, 6, 1989.

[7] 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.

[8] T. Biedl, F.J. Brandenburg et X. Deng. Crossings and permutations. *LNCS*, 3843: 1–12, 2005.

[9] Guillaume Blin, Maxime Crochemore, Sylvie Hamel et Stéphane Vialette. Median of an odd number of permutations. *Pure Mathematics and Applications*, 21(2), 2011.

[10] Borda. Mémoire sur les élections au scrutin. *Histoire de l'Académie Royale des Sciences*, 1781.

[11] Sergey Brin et Larry Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30, 1998.

[12] Guillaume Blin Alain Denise Sarah Cohen-Boulakia Bryan Brancotte, Bo Yang et Sylvie Hamel. Rank aggregation with ties : Experiments and analysis. Dans *Proceedings of the VLDB Endowment (PVLDB)*, 2015.

[13] Kobylanski P. Chanas, S. A new heuristic algorithm solving the linear ordering problem. *Computational Optimization and Applications*, 6, 1996.

[14] Sarah Cohen-Boulakia, Alain Denise et Sylvie Hamel. Using medians to generate consensus rankings for biological data. *Lecture Notes in Computer Science*, 6809 (1), 2011.

[15] Condorcet. Essai sur l'application de l'analyse à la probabilité des décisions rendue à la pluralité des voix. *Paris : Imprimerie royale*, 1785.

[16] Davenport A. Kalagnanam J. Conitzer, V. Improved bounds for computing kemeny rankings. Dans *Proceedings of the 21st National Conference on Artificial Intelligence*, pages 620–627, 2006.

[17] A. H. Copeland. A reasonable social welfare function. *University of Michigan Seminar on Applications of Mathematics to the social sciences*, 1951.

[18] Kalagnanam J. Davenport, A. A computational study of the kemeny rule for preference aggregation. Dans *Proceedings of the 19th National Conference on Artificial Intelligence*, pages 697–702, 2004.

[19] 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), 2006.

[20] C. Dwork, R. Kumar, M. Naor et D. Sivakumar. Rank aggregation methods for the web. *in proceedings of the 10th WWW*, pages 613–622, 2001.

[21] H. Spakowski E. Hemaspaandra et J. Vogel. The complexity of kemeny elections. *Theoretical Computer Science*, 349, 2005.

[22] Sylvie Hamel et Robin Milosz. Medians of permutations : when A = M(A). Dans *Procceedings of 12th International Permutation Patterns Conference*, pages 68–71, 2014.

[23] R.M Karp. Reducibility among combinatorial problems. *Complexity of Computer Communications*, 1972.

[24] M. Karpinski et W. Schudy. Faster algorithms for feedback arc set tournament, kemeny rank aggregation and betweenness tournament. *LNCS*, 6506:3–14, 2010.

[25] J. Kemeny. Mathematics without numbers. *Daedalus*, 88:577–591, 1959.

[26] M. Kendall. A new measure of rank correlation. *Biometrika*, 30:81–89, 1938.

[27] C. Kenyon-Mathieu et W. Schudy. How to rank with few errors. *STOC07*, pages 95–103, 2007.

[28] Phadnis K. Patterson A. Bilmes J. Meila, M. Consensus ranking under the exponential model. Dans *Proceedings of the 23rd Annual Conference on Uncertainty in Artificial Intelligence*, pages 285–294, 2007.

[29] Robin Milosz et Sylvie Hamel. Medians of permutations : building constraints. Dans *Proceedings of 2nd Conference on Algorithms and Discrete Applied Mathematics*, 2016.

[30] Robin Milosz et Sylvie Hamel. Heuristic, brancha-and-bound solver and improved space reduction for the median of permutations problem. Dans *Proceedings of International Workshop On Combinatorial Algorithms*, 2017.

[31] J. Guo R. Niedermeier N. Betzler, M. R. Fellows et F. A. Rosa-mond. Fixed-parameter algorithms for kemeny scores. Dans *Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management*, pages 60–71, 2008.

[32] J. Guo R. Niedermeier N. Betzler, M. R. Fellows et F. A. Rosamond. How similarity helps to efficiently compute kemeny rankings. Dans *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems*, 2009.

[33] N. Nishimura et N. Simjour. Parameterized enumeration of (locally-) optimal aggregations. *IWADS13, LNCS*, 8037:512–523, 2013.

[34] Larry Page. Pagerank : Bringing order to the web. *Stanford Digital Library Project, talk*, 1997.

[35] Frans Schalekamp et Anke van Zuylen. Rank aggregation : Together we're strong. Dans *Proceedings of the 11th SIAM Workshop on Algorithm Engineering and Experiments, ALENEX*, pages 38–51, 2009.

[36] J. Sese et S. Morishita. Rank aggregation method for biological databases. *Genome informatic series*, pages 506–507, 2001.

[37] N. Simjour. Improved parameterized algorithms for the kemeny aggregation problem. *IWPEC09, LNCS*, 5917:312–323, 2009.

[38] J. Starlinger, B. Brancotte, S. Cohen-Boulakia et U. Leser. Similarity search for scientific workflows. *Proceedings of the VLDB Endowment*, 7(12), 2014.

[39] M. Truchon. An extension of the condorcet criterion and kemeny orders. *Internal Report, Université Laval*, page 16 pages, 1998.

[40] 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.

[41] H.S. Wilf. Generatingfunctionology,. *Academic Press, NY*, 1990.

[42] Levenglick A. Young, H.P. A consistent extension of condorcet's election principle. *SIAM Journal on Applied Mathematics*, C35, 1978.

[43] Peyton Young. Condorcet's theory of voting. *Math. Inform. Sci. Humaines*, 111, 1990.

[44] Peyton Young. Optimal voting rules. *Journal of Economic Perspectives*, 9, 1995.

[45] Gunter M. Ziegler. *Graduate Texts in Mathematics - vol 152 : Lectures on Polytopes*. Springer-Verlag New York, 1995.

Annexe I

Article de IWOCA017

Figures, graphiques et tableaux complémentaires à l'article de IWOCA017

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,
{robin.milosz, sylvie.hamel}@umontreal.ca

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.

1 Introduction

The problem of finding medians of a set of m permutations of $\{1, 2, \ldots, n\}$ under the Kendall-τ distance [13], often cited as the Kemeny Score Problem [12] 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 [8], then corrected in [4]), 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 [1, 14, 21] or fixed-parameter ones [3, 11, 20]. Other theoretical approaches aiming at reducing the search space for this problem have also been developed [2, 5–7, 19].

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 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 [6, 19] complemented by our simulated annealing heuristic, thus reducing significantly the search space for this median. Third, we present, in Section 5, an implementation of an exact solver: a branch-and-bound algorithm that is powered by the two previous approaches. Finally, Section 6 gives some thoughts on future 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)

2 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 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:

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

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 metaheuristic for locating a good approximation to the global optimum of a given function in a large search space [15]. It is an adaptation of the Metropolis-Hasting algorithm [16] 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 [9]. 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 $d_{KT}(neighbour(\pi)), \mathcal{A}) - d_{KT}(\pi, \mathcal{A})$. 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].

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 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

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

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: $t_0 \longleftarrow (0.25m + 4.0)n$
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.

4 Space reduction technique

In [18] and [19] 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.* ([6]), 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 [18], 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 multiset of elements between i and j, that we will called **interference multiset**, act as

[2] More detailed explanations and some examples for these Major Order Theorems can be found in Section 4 of [18].

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 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 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 [19], 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 1.

4.2 A new lower bound

Given a set of permutations $\mathcal{A} \subseteq \mathbb{S}_n$, Davenport and Kalagnanam [7] 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 [10] 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 [6], 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 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. 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 1, 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.

	$n = 15$				$n = 20$				$n = 30$				$n = 45$			
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

Table 1. 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 $n = 45$.

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.

5 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 3 with the constraints obtained in Section 4.4 to avoid exploring not promising search subtrees.

5.1 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 3 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 5.2 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 $d_{KT}(\pi_{leaf}, \mathcal{A}) = Ub_{\mathcal{A}}$, 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.

5.2 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 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 $\mathcal{A}$.

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, 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 [17], 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.

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-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.

7 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.

References

1. N. Ailon, M. Charikar and N. Newman, *Aggregating inconsistent information: ranking and clustering*, Journal of the ACM, 55(5), pp.1–27, 2008.
2. N. Betzler, R. Bredereck, R. Niedermeier, *Theoretical and empirical evaluation of data reduction for exact Kemeny Rank Aggregation*, Autonomous Agents and Multi-Agent Systems, vol. 28, pp.721–748, 2014.
3. N. Betzler *and al.*, *Average parameterization and partial kernelization for computing medians.*, Journal of Computer and System Sciences, 77(4), pp. 774–789, 2011.
4. T. Biedl, F.J. Brandenburg and X. Deng, *Crossings and Permutations*, LNCS 3843, pp. 1–12, 2005.
5. G. Blin, M. Crochemore, S. Hamel and S. Vialette, *Median of an odd number of permutations*, Pure Mathematics and Applications, 21 (2), pp. 161–175, 2011.
6. V. Conitzer, A. Davenport, and J. Kalagnanam, *Improved bounds for computing Kemeny rankings*, Proceedings of the 21st conference on Artificial intelligence - Volume 1, AAAI'06, pp. 620–626, 2006.
7. A. Davenport and J. Kalagnanam, *A Computational Study of the Kemeny Rule for Preference Aggregation*, Proceedings of the 19th National Conference on Artificial Intelligence, AAAI'04, pp.697–702, 2004.
8. C. Dwork, R. Kumar, M. Naor and D. Sivakumar, *Rank Aggregation Methods for the Web*, in proceedings of the 10th WWW, pp.613-622, 2001.
9. R.A. Fisher, F. Yates, *Statistical tables for biological, agricultural and medical research*, (3rd ed.). London: Oliver & Boyd. pp. 2627, 1948.
10. R.M. Karp, *Reducibility Among Combinatorial Problems*, Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, pp. 85–103, 1972.
11. M. Karpinski and W. Schudy, *Faster algorithms for feedback arc set tournament, Kemeny rank aggregation and betweenness tournament*, LNCS 6506, pp. 3–14, 2010.
12. J. Kemeny, *Mathematics without numbers*, Daedalus 88 , pp. 577591, 1959.
13. M. Kendall, *A New Measure of Rank Correlation*, Biometrika, 30, pp.81-89, 1938.
14. C. Kenyon-Mathieu and W. Schudy, *How to rank with few errors*, STOC'07, pp. 95-103, 2007.
15. S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, *Optimization by Simulated Annealing*, Science, 220 (4598), pp.671–680, 1983.
16. N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, N. Marshall, A. H. Teller, E. Teller, *Equation of State Calculations by Fast Computing Machines*, The Journal of Chemical Physics, 21-6, pp. 1087–1092, 1953.
17. A. Ali, M. Meilă, *Experiments with Kemeny ranking: What works when?*, Mathematical Social Science, 64, pp. 28–40, 2012.
18. R. Milosz and S. Hamel, *Medians of Permutations: Building Constraints*, LNCS 9602, Second International Conference on Algorithms and Discrete Applied Mathematics, pp. 264–276, 2016.
19. R. Milosz and S. Hamel, *Space reduction constraints for the median of permutations problem*, submitted to Journal of Discrete Applied Mathematics.
20. N. Nishimura and N. Simjour, *Parameterized enumeration of (locally-) optimal aggregations*, IWADS13, LNCS 8037, pp.512–523, 2013.
21. A. vanZuylen and D.P. Williamson, *Deterministic pivoting algorithms for constrained ranking and clustering problems*, Mathematics of Operations Research, 34(3), pp.594–620, 2009.

Appendix A: Simulated Annealing

A.I: Pseudo-code

The pseudo-code of our simulated annealing heuristic is given in Algorithm 1:

Algorithm 1: Simulated Annealing($\mathcal{A}, nbRuns, nbMvts, \alpha$)

Data: a set $\mathcal{A}$ 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 $d_{KT}(\pi, \mathcal{A})$

```
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
    for i ← 1 to nbRuns do
        T ⟵ (0.25m + 4.0)n      // beginning temperature of the system
        π ⟵ RandomPermutation(n)      // random starting point
        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 BestM with 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
```

A. II: Choosing the parameters

For the parameters that were not discuss in Section 3.2, here is more information on why and how they were chosen.

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 [15] and suits well our problem in practice. 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 ($t_n = \alpha t_0/n$).

Initial temperature: The initial temperature of the system was chosen in a way that the worst possible circular movement would be accepted with probability $p \sim 37\%$; 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).

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. $[\Delta_{max}(m, n) - (0.25 \times m + 4.0) \times n]/[\Delta_{max}(m, n)]$

As the maximum rarely occurs (probably in conditions having the current solution close to the median and a neighbour produced by an extreme circular 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 1 and 2 that experimentations did validate this choice.

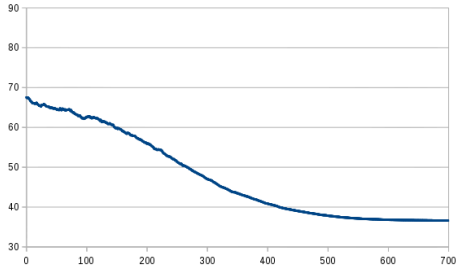

Fig. 1. The average distance of a SA solution per iteration, where the x-axis represent the number of iterations and the y-axis the average distance. We generated 1000 different sets $\mathcal{A}$ of $m = 3$ permutations of $[n]$, $n = 10$. For each instance, we ran the SA heuristic and noted at each 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 2. 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.

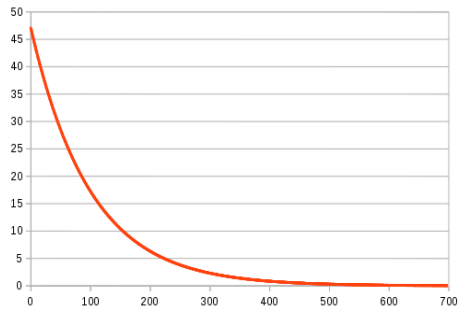

Fig. 2. SA temperature of a set $\mathcal{A}$ of $m = 3$ permutaitions of $[n]$, $n = 10$, using the exponential cooling scheme with initial temperature set to $(m \times 0.25 + 4) \times n = 47.5$ and $\alpha = 0.99$ cooling constant.

Some extensive testing: 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 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 the optimal score for each of them, 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 the 95% movements threshold $t95$ such as 95% of successful runs required less movements than this threshold.

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 3 and Tables 3 to 6 show the results of our experimentation.

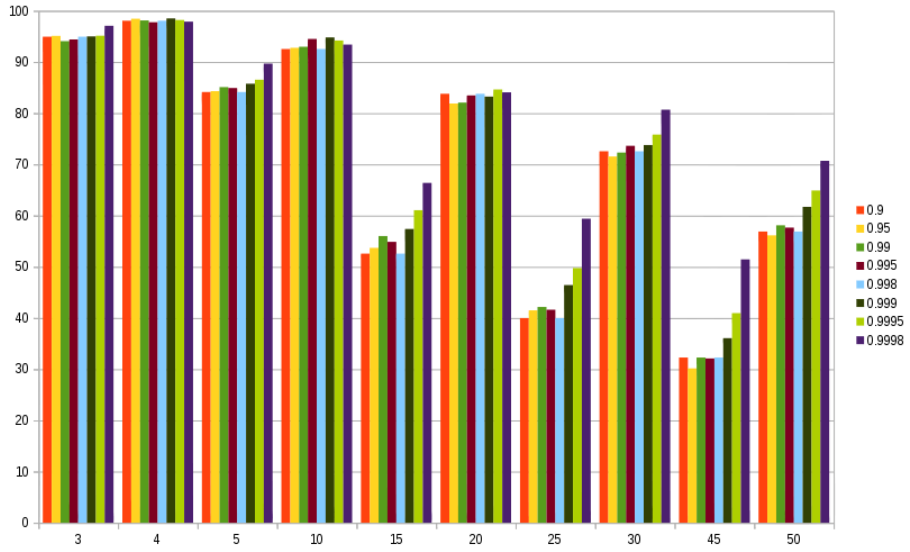

Fig. 3. Average probability of success of SA for each α, $n = 38$ and different m with the number of movements of SA limited to 100000. Based on 500 randomly generated sets $\mathcal{A}$ for each m, and 1000 SA runs per set. We can observe that the cooling constant has limited impact on the probability of success for $m = 3, 4$, but greatly affects bigger m.

α/n	10	14	16	20	24	28	32	38
0.9	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.

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.

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) 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 3), 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:

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

Number of movements: Our formulas for $nbMvts$, the number of movements allowed in our SA heuristic, where 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).

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

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.

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.

Number of SA runs: Finally, the number $nbRuns$, of repetitions of SA was determined so that the average probability of success is at least 99.9% for each (m, n) in the considered range. Let p_{t95} be the probability of success with $nbMvts$ movements. The probability of failure is $f = 1 - p_{t95}$. If we run SA $nbRuns$ of times, the probability of failure will be f^{nbRuns}. As we want a probability of failure lower than 0.001, we choose 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).

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

Appendix B: Lower-upper bound constraints method

B.I: Pseudo-code

The pseudo-code of our MOT+LUBC method is given in Algorithms 2 and 3 below:

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 ← ∅
    improved ← True
    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)}
                    improved ← True
                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)}
                        improved ← True
                    end if
                end for
            end for
        end if
    end while
    return C
end
```

Algorithm 3: LUBC($(\mathcal{A}, C, i, j, UpperBound)$)

Data: a set $\mathcal{A}$ of permutations, a set C of valid constraints, first element i, second element j, an known upper bound $UpperBound$
Result: If $i \prec j$ is a valid constraint
begin
  $isValid \leftarrow False$
  $n \longleftarrow$ number of elements in the permutations of $\mathcal{A}$
  $LowerBound \leftarrow 0$
  $C' \leftarrow C \ \cup (i, j)$
  $C' \leftarrow transitive\_closure(C')$
  $V = \{1, 2, 3, ..., n-1, n\}$
  $E = \{(i, j) | i \in V, j \in V, i \neq j\}$
  $G = (E, V)$
  for $(i, j) \in E$ **do**
    $w(i, j) = R_{ji} - min\{R_{ij}, R_{ji}\}$
    $LowerBound \leftarrow LowerBound + min\{R_{ij}, R_{ji}\}$
  end for
  for $(i, j) \in C'$ **do**
    $LowerBound \leftarrow LowerBound + w(j, i)$
    $w(j, i) = 0$
  end for
  for $i \leftarrow 1$ **to** n **do**
    for $j \leftarrow 1$ **to** n **do**
      for $k \leftarrow 1$ **to** n **do**
        $min \leftarrow min\{w(i, j), w(j, k), w(k, i)\}$
        if $min > 0$ **then**
          $w(i, j) \leftarrow w(i, j) - min$
          $w(j, k) \leftarrow w(j, k) - min$
          $w(k, i) \leftarrow w(k, i) - min$
          $LowerBound \leftarrow LowerBound + min$
        end if
      end for
    end for
  end for
  if $LowerBound > UpperBound$ **then**
    $isValid \leftarrow True$
  end if
  return $isValid$
end

B.II: Some more results

Here are tables presenting more extensive results for the MOT, MOT+LUBC, MOTe and MOTe+LUBC methods.

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

Table 9. Average efficiency of MOT 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 number of sets generated for each n is given in Table 13

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

Table 10. Average efficiency of MOT+LUBC constraints in % of resolution of pairs of elements on the same sets of permutations generated in Table 9.

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

Table 11. Average efficiency of 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 number of sets generated for each n is given in Table 13

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

Table 12. Average efficiency of MOTe+LUBC constraints in % of resolution of pairs of elements on the same sets of permutations generated in Table 11.

n	10	15	20	25	30	35	40	45
nbInstances	100000	80000	50000	20000	10000	4000	2000	1000

Table 13. Number of uniformly distributed random sets generated for each pair (m, n) for the statistics presented in Tables 9 10 11 12.

B.III: Time complexity of the new MOT+LUBC and MOTe+LUBC approaches

As the complexity of the MOT method is of $O(n^2m + n^3 + n^2k)$, the complexity of the LUBC method is $O(n^2m + n^3 + n^5k)$, since the lower bound method in $O(n^3)$ is repeated on all pairs of elements over k iterations (number of time we find new constraints). We observed in practice (see Figure 4) that the time required to compute the new constraints method is amortized in $O(n^3)$. We also observed that the average number of iteration 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.

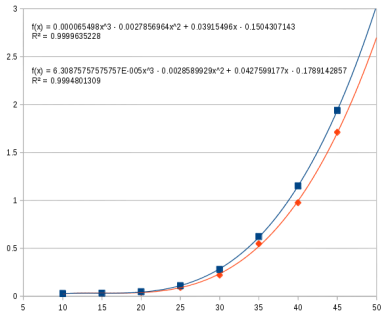

Fig. 4. Average time in seconds to compute the new constraints methods on one instance of (m, n) with $m \in \{3, ..., 50\}$ for a specific n. In blue, the MOT+LUBC time and in orange, the MOTe+LUBC time. Average calculated over 1000-100000 instances per (m, n).

Polynomial regression for the MOT+LUBC gives us an approximation of the time (in seconds): $time(n) = 0.0000655n^3 - 0.002786n^2 + 0.03915n - 0.15043$. 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 $m \leq 50$.

Appendix C: Pseudo-code of our BnB method

The pseudo-code of our BnB method is given in Algorithm 4 below:

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 $L = \emptyset$ **then**
    $\pi \leftarrow$ permutation associated to x
    if $d_{KT}(\pi, \mathcal{A}) < UpperBound$ **then**
      $M \leftarrow \{\pi\}$
      $UpperBound \leftarrow d_{KT}(\pi, \mathcal{A})$
    else if $d_{KT}(\pi, \mathcal{A}) = UpperBound$ **then**
      $M \leftarrow M \cup \{\pi\}$
    end if
  else
    for $i \leftarrow 1$ **to** $|L|$ **do**
      $x_{k+1} \longleftarrow l_i$
      $isAccepted \leftarrow True$
      $x' \leftarrow (x_1 x_2, \ldots, x_k, x_{k+1}, 0, \ldots, 0)$
      $L' \leftarrow L \setminus \{x_{k+1}\}$
      $b(x') \leftarrow b(x)$
      $b(x')_{x_{k+1}} \leftarrow 1$
      $kd(x') = \sum_{i=1}^{|x'|} \sum_{j=i+1}^{|x'|} R_{x'_i x'_j} + \sum_{i=1}^{|x'|} \sum_{j=1}^{|L'|} R_{x'_i l'_j}$
      $lb(L') = \sum_{i=1}^{|L'|} \sum_{j=i+1}^{|L'|} (min\{R_{l'_i l'_j}, R_{l'_i l'_j}\}) + tri(L')$
      if $x_k \prec x_{k+1}$ *is the minor order* **then**
        $isAccepted \leftarrow False$
      end if
      if $x_{k-1} \prec x_k \prec x_{k+1}$ *is not optimal* **then**
        $isAccepted \leftarrow False$
      end if
      if $\exists l_i \in L' \mid C_{l_i, x_{k+1}} = 1$ **then**
        $isAccepted \leftarrow False$
      end if
      if $kd(x') + lb(L') > UpperBound$ **then**
        $isAccepted \leftarrow False$
      end if
      if $< b(x'), v > \in TopScores$ *(for any v)* **then**
        if $kd(x') > v$ **then**
          $isAccepted \leftarrow False$
        else if $kd(x') < v$ **then**
          $TopScores \leftarrow TopScores \setminus < b(x'), v >$
          $TopScores \leftarrow TopScores \cup \{< b(x'), kd(x') >\}$
        end if
      else
        $TopScores \leftarrow TopScores \cup \{< b(x'), kd(x') >\}$
      end if
      if $isAccepted$ **then**
        Branch-and-Bound($\mathcal{A}, x', L'$)
      end if
    end for
  end if
end