Agrégation de classements avec égalités : algorithmes, guides à l'utilisateur et applications aux données biologiques

Bryan Brancotte

N° ordre : 42

Université Paris-Sud

Ecole Doctorale 427 :
Informatique Paris-Sud

Laboratoire :
Laboratoire de Recherche en Informatique (LRI)

Discipline : Informatique

Thèse de doctorat

Soutenue le 25 septembre 2015 par

Bryan Brancotte

Agrégation de classements avec égalités : algorithmes, guides à l'utilisateur et applications aux données biologiques

Directeurs de thèse :	Mr Alain Denise	Professeur des universités *LRI, Université Paris-Sud*
	Mme Sarah Cohen-Boulakia	Maître de conférences HDR *LRI, Université Paris-Sud*
Composition du jury :		
Rapporteurs :	Mr Bernd Amann	Professeur des universités *LIP6, Université Pierre et Marie Curie*
	Mr Stephane Vialette	Directeur de recherche *LIGM Université Paris-Est Marne-la-Vallée*
Examinateurs :	Mr Khalid Belhajjame	Maître de conférences *LAMSADE, Université Paris-Dauphine*
	Mme Nicole Bidoit	Professeur des universités *LRI, Université Paris-Sud*
	Mr Ulf Leser	Professeur des universités *WBI, Université Humböldt, Berlin*

Table des matières

Introduction

Cette dernière décennie, nous sommes entrés dans une ère de profusion d'information, une profusion telle qu'elle tend à submerger celui qui l'explore, c'est l'ère du *Big Data*. Si la mission Rosetta nous a rappelé la difficulté qu'il y a à obtenir certaines données, d'autres domaines tels que le séquençage à haut débit, le grand collisionneur de hadrons (LHC), mais aussi les géants du Web nous rappellent tous les jours la difficulté qu'il y a à explorer les masses de données disponibles pour y trouver l'information pertinente qui permettra d'identifier le site web répondant au mieux à une question, le gène qui cause une maladie, la publication qui donnera l'intuition pour une avancée scientifique. De nombreuses approches proposent de fouiller les masses de données pour en extraire et trier les données les plus pertinentes. Deux problèmes se posent alors. D'abord, différentes approches similaires appliquées au même ensemble de données peuvent extraire des données différentes mais dont on souhaite exploiter la complémentarité. Se pose alors le problème d'agréger les ensembles de données résultats en un unique résultat. Ensuite, même si ces ensembles de données extraits sont plus petits que l'ensemble de départ, leur cardinalité est trop grande pour que chaque donnée puisse être inspectée et étudiée individuellement. Il est donc particulièrement important de classer l'ensemble de ces données. Se pose alors le problème d'agréger les données obtenues par les différentes approches en exploitant les classements fait par chaque approche.

On est alors dans le contexte de L'AGRÉGATION DE CLASSEMENTS.

Dans le domaine biomédical, l'évolution des techniques de séquençage à haut débit a créé une avalanche de données qu'il nous faut traiter, analyser et explorer. En passant de 10 ans pour séquencer le premier génome humain, à bientôt quelques heures, cette avalanche de données a permis l'apparition d'un myriade de sources de données biologiques (1621 selon [GRFS15], dont 189 sont consacrées uniquement aux gènes et maladies chez l'Homme). Ces sources de données biologiques ont des centres d'intérêts différents, sont certaines fois redondantes, et utilisent des formats différents. Néanmoins, les sources de données biologiques publiques ne sont pas indépendantes les unes des autres mais liées par des liens de références croisées qui permettent d'accéder à des données complémentaires dans d'autres sources.

L'exploration de ces sources pour y trouver l'information pertinente a fait l'objet de nombreux travaux [BY06 ; CB+07 ; SIY06 ; Det+09 ; Ras+06 ; Var+09]. L'un d'eux, BioGuideSRS[CB+07], a proposé une exploration des sources de données en se basant sur des chemins d'exploration des sources de données. BioGuideSRS a montré que les informations étaient trouvées, mais qu'il était nécessaire de trier les résultats. Dans le cadre de ce projet, nous avons alors développé [Bra09 ; Bra12] plusieurs approches de tri en se basant sur des algorithmes connus comme PageRank [Pag+99], et des variantes [Din+02 ; Kle99 ; BHP04 ; RD01], mais aussi des algorithmes plus simples comme le classement des résultats en fonction du nombre de chemins permettant d'y accéder. Ces approches de tri ont mis en avant le fait qu'elles pouvaient être complémentaires, et qu'il y avait un réel besoin de trouver un consensus entre les diverses réponses proposées par chaque approche de tri.

On retrouve le besoin de construire des consensus dans d'autres domaines, en particulier dans la mise en place de systèmes de votes capable de prendre en compte des points de vue différents mais complémentaires. Les premiers travaux dans ce domaine remontent alors au siècle des lumières où se sont bâties les prémisses des démocraties modernes. Les travaux de J-C de Borda (1781) sont particulièrement intéressants : il montra qu'avec un système d'élection classique (une voix par électeur) le candidat élu pouvait ne pas refléter les préférences des électeurs. Il proposa alors que chaque électeur exprime son vote en établissant une liste dans laquelle il ordonne l'ensemble des candidats. Le résultat des suffrages est alors lui aussi une liste. La problématique du consensus entre listes est alors très présente. Il est à noter que cette problématique de consensus entre listes établies par des personnes se retrouve aussi lorsque des experts ordonnent des objets d'étude, et qu'il faut ensuite calculer un consensus entre ces listes.

Le besoin de produire des consensus entre listes d'éléments est donc présent dans des domaines particulièrement nombreux et divers. Il est important de noter que le problème de l'agrégation de classements est NP-difficile [Dwo+01], et qu'y répondre est donc un défi. Nous nous inscrivons dans la perspective de répondre à ce besoin, en analysant les distances et pré-traitements permettant de préciser le besoin et l'interprétation des données, et ainsi en proposant une vue haut niveau de ces distances et prétraitements. Face à la difficulté du calcul d'un consensus exact, de nombreuses approximations et heuristiques ont été développées. Face à un ensemble de données dont on souhaite faire le consensus, choisir l'algorithme adapté est devenu une tache complexe. Nous nous inscrivons aussi dans la perspective de répondre au besoin de consensus en évaluant les algorithmes existants sur des données de natures différentes (synthétiques ou réelles, similaires ou non), et en établissant des recommandations quant aux algorithmes à utiliser en fonction de

situation réelles.

Lorsque l'on agrège des listes pour produire un consensus, un second besoin tend à apparaître, c'est celui de considérer que les éléments peuvent être à égalité dans une liste. On parlera alors de classement. Même si des moteurs de recherche tel que nous les utilisons n'indiquent pas les scores obtenus par les différentes pages qu'ils affichent, ne permettant donc pas de voir si certaines sont à égalité ou non, dans d'autres domaines l'égalité dans le classement est explicite. C'est le cas notamment des données biologiques. Plusieurs gènes impliqués dans un processus biologique peuvent en effet être de pertinence égale et doivent être considérés comme tel. Si l'on considère le consensus entre listes d'éléments établies par des experts, l'égalité entre éléments fait partie intégrante de l'opinion de l'expert, et doit donc nécessairement être considérée.

La prise en compte des égalités est un besoin fort que nous adressons dans ce mémoire. Peu d'algorithmes permettant de considérer les égalités, nous adapterons alors plusieurs algorithmes du domaine afin de répondre à cette problématique. Ces adaptations seront évaluées dans une large étude expérimentale permettant d'identifier les caractéristiques dans les données qui ont une influence sur la qualité des résultats d'un algorithme. Cette étude aboutira à la production des recommandations.

Les masses de données explorées sont telles qu'un outil qui retourne une liste d'informations, d'éléments pertinents, ne peut pas couvrir l'ensemble des données possibles. En d'autres termes, les outils vont proposer à un utilisateur un classement sur un sous-ensemble des résultats possibles. Dans le cas d'un consensus entre personnes, il est admissible que les électeurs soient obligés d'ordonner l'ensemble des candidats, il n'est cependant pas acceptable d'obliger un expert à ordonner l'ensemble des éléments qui lui sont proposés.

Les classements qui sont agrégés pour produire un consensus ne sont donc, dans des cas réels, pas nécessairement sur les mêmes éléments, on qualifiera ces classements d'incomplets. Nous étudions dans ce mémoire, à la fois algorithmiquement et expérimentalement, différentes approches (distances et pré-traitements) pour prendre en compte ces classements.

Afin de replacer dans une vue d'ensemble les besoins qui motivent l'utilisation d'algorithmes pour calculer des consensus, nous caractériserons ces besoins, amenant ainsi à un *guide à l'utilisateur* pour la préparation des données et le choix

du formalisme dans lequel se placer pour calculer un consensus par le choix des distances et pré-traitements adaptés.

La première contribution de nos travaux est la proposition d'un guide à l'utilisateur permettant de caractériser le besoin d'un utilisateur et de le guider dans le choix à la fois d'un pré-traitement de ses données mais aussi d'un formalisme dans lequel calculer un consensus, via la distance à utiliser. Cette caractérisation faite, nous adaptons ou proposons de nouveaux algorithmes [Bra+15] pour calculer des consensus en prenant en compte les égalités et les classements incomplets, et proposons des résultats quant à la complexité du problème de l'agrégation de classements [BM15] dans ces cas.

La seconde contribution de nos travaux [Bra+15] est une évaluation des algorithmes sur des données réelles et synthétiques visant à reproduire des caractéristiques qu'ont les données réelles telles que la similarité entre classements, ou encore la présence d'égalités et les pré-traitements appliqués aux données. Cette large évaluation passe par la proposition d'une nouvelle méthode pour générer des données synthétiques avec similarités. L'aboutissement de cette évaluation est alors un ensemble de recommandations sur le choix de l'algorithme à utiliser en fonction des données et des besoins de l'utilisateur.

La troisième contribution est l'application des connaissances présentes dans ce mémoire et se décompose en trois axes. Dans un premier temps, nous avons conçu ConQuR-Bio [Bra+14], un outil disponible en ligne reformulant *à-la-volé* des requêtes textuelles d'utilisateur en nous fondant sur des terminologies biomédicales, pour ensuite proposer un consensus des résultats obtenus pour chaque reformulation. Dans un second temps, nous avons produit des consensus entre experts [Sta+14]. Ces derniers ont classé par rapport à des objets de référence un ensemble d'objets candidats, tout en acceptant les égalités, mais aussi le fait que tous les objets ne sont pas nécessairement classés. Dans un troisième temps, nous avons proposé Rank'n'ties, une plateforme disponible en ligne permettant de reproduire les analyses faites dans [Bra+15], et de les étendre à de nouveaux algorithmes.

Le mémoire s'organise de la façon suivante :

- Le Chapitre 1 définit formellement les concepts fondamentaux relatifs au problème l'agrégation de classements : les classements, les distances, les mesures, et finalement les notations utilisées dans le cadre de ce problème.
- Le Chapitre 2 dresse un état de l'art des algorithmes permettant d'agréger des classements, mais aussi des approches pour travailler avec des classements incomplets, et finalement synthétise les résultats obtenus dans la littérature.

- Le Chapitre 3 introduit le guide à l'utilisateur permettant de caractériser le besoin de calcul de consensus. Il introduit ensuite les évolutions des algorithmes pour prendre en compte une problématique majeure : les égalités entre éléments. Ce chapitre introduit aussi un nouvel algorithme exact et des améliorations d'un heuristique pour le calcul de consensus avec égalités. Finalement, ce chapitre étudie la complexité du problème de l'agrégation de classements en prenant en comptes les égalités ou les classements incomplets.
- Le Chapitre 4 introduit le cadre d'évaluation, les méthodes de génération de données et les données utilisées pour évaluer les algorithmes. L'évaluation est conduite dans un second temps, pour finalement produire un ensemble de recommandations sur le choix d'un algorithme.
- Le Chapitre 5 présente finalement deux applications majeures des travaux de ce manuscrit. Il s'agit dans un premier temps de ConQuR-Bio une application reformulant les requêtes biomédicales d'utilisateur pour proposer un consensus entre les classements de résultats. Il s'agit dans un second temps de l'application au consensus entre experts.

Enfin nous concluons et formulons des perspectives découlant des travaux de ce mémoire. On notera que chaque chapitre se termine par un résumé reprenant les points et contributions importantes du chapitre.

Chapitre 1

Définition du problème

Sommaire

1.1 Introduction

Le problème de l'agrégation de classements est, à partir d'un ensemble de classements fournis en entrée, de trouver un classement qui va être un consensus entre ces classements. Ce classement est décrit comme un consensus car il va tendre à minimiser une distance entre lui-même et les classements en entrée. Selon les communautés, le problème a de nombreux noms : *problème d'agrégation de rang de Kemeny* (*Kemeny rank aggregation problem*) [CDK06 ; Bet+11 ; AM12 ; BBN13], *problème du classement consensuel* (*consensus ranking problem*) [Mei+12], *problème du classement médian* (*median ranking problem*) [CBDH11], ou encore *problème d'agrégation de préférences* (*preference aggregation problem*) [DK04]. Certaines approches considèrent le fait que, dans un classement, des éléments peuvent être *ex-æquo* [Fag+06 ; CBDH11 ; Bra+14], mais cela reste marginal comparé aux

cas où le but recherché est d'avoir un classement sans égalité [Bor81 ; Dwo+01 ; DK04 ; CDK06 ; ACN08 ; SZ09 ; Bet+11 ; AM12 ; Mei+12 ; BBN13].

Dans ce chapitre nous définirons (*cf.* Section 1.2) ce qu'est un classement, une distance entre classements et quelles distances ont été proposées, nous formaliserons ensuite le problème de l'agrégation de tris, enfin nous aborderons la NP-Complétude du problème. Dans un second temps (*cf.* Section 1.3), nous prendrons en compte le fait d'avoir des éléments *ex-æquo*, on parlera alors de *classements avec égalités*. Après avoir formalisé ce qu'est un classement avec égalités et un jeu de données, nous étudierons les distances proposées et adaptées à la prise en compte des égalités, et définirons alors le problème de l'agrégation de classements formellement. Dans un troisième temps (*cf.* Section 1.4) nous formaliserons ce qu'est un classement et un jeu de données incomplet pour finalement formaliser des notations qui seront utiles pour la suite de ce mémoire.

1.2 Agrégation de classements

Le problème de l'agrégation de classements a initialement été défini pour des classements sans égalité, que l'on appelle alors des permutations.

Définition 1.1. Une permutation π est une bijection de l'ensemble $[n] = \{1, 2, ..., n\}$ sur lui-même.

L'ensemble des permutations de $[n]$ est noté $\mathbb{S}_n$, et sa taille est $|\mathbb{S}_n| = n!$. On considérera que $\pi[x]$ dénote la position de l'élément x dans la permutation π.

Un classement pouvant contenir des entiers, mais aussi des compétiteurs ou encore gènes, on considère alors qu'il existe toujours une fonction bijective (certaines fois implicite) permettant de ramener un ensemble de n éléments à l'ensemble $[n]$. Pour plus de lisibilité les exemples seront construit avec des lettres, la fonction bijective ramenant une lettre à un entier considère alors la position de la lettre dans l'alphabet : $A \equiv 1, B \equiv 2, \ldots$. On considérera alors que $A < B$ car $A \equiv 1 < 2 \equiv B$.

Exemple 1.1. Soit une permutation $\pi = [A, B, C]$, l'élément A est le premier suivi de B, et terminé par C. On dit que A est au rang 1, ou $\pi[A] = 1$ suivant la notation précédemment introduite.

1.2.1 Mesures de dissimilarité

Afin de comparer deux permutations entre elles, plusieurs mesures de dissimilarité ont été proposées. Elles ont pour point commun de se concentrer sur les désaccords quant à la position, ou le rang, qu'ont les éléments deux à deux. Certaines mesures sont des métriques (*cf.* Définition 1.2), d'autre des pseudo-métriques (*cf.* Définition 1.3), d'autre encore des semi-métriques (*cf.* Définition 1.4).

Définition 1.2. Une métrique, ou distance, est une fonction d de $\mathbb{S}_n \times \mathbb{S}_n \to \mathbb{R}$ qui satisfait trois conditions :

1. Identité : $d(x, y) = 0 \iff x = y$;
2. Symétrie : $d(x, y) = d(y, x)$
3. Inégalité triangulaire. : $d(x, y) + d(y, z) \geq d(x, z)$

Définition 1.3. En se basant sur la définition 1.2, une pseudo-métrique [SSS78] est une fonction d de $\mathbb{S}_n \times \mathbb{S}_n \to \mathbb{R}$ qui satisfait les contraintes de symétrie (2) et d'inégalité triangulaire (3).

Définition 1.4. En se basant sur la définition 1.2, une semi-métrique [SSS78] est une fonction d de $\mathbb{S}_n \times \mathbb{S}_n \to \mathbb{R}^+$ qui satisfait les contraintes d'identité (1) et de symétrie (2).

1.2.1.1 Distance de Kendall-τ

Une mesure très classiquement utilisée [Dwo+01 ; CDK06 ; SZ09 ; Ail10 ; AM12 ; Mei+12 ; BBN13] pour comparer deux permutations est la distance de Kendall-τ [Ken38]. Elle compte le nombre de paires d'éléments inversés entre deux permutations.

Définition 1.5. La distance de Kendall-τ, notée ici D, est définie entre deux permutations $\pi \in \mathbb{S}_n$ et $\sigma \in \mathbb{S}_n$ de la façon suivante :

$$D(\pi, \sigma) = |\{(i, j) : i < j \wedge (\pi[i] < \pi[j] \wedge \sigma[i] > \sigma[j] \vee \pi[i] > \pi[j] \wedge \sigma[i] < \sigma[j])\}| \tag{1.1}$$

Cette distance est parfois appelée **distance du tri à bulle** [Dwo+01] car elle est égale au nombre de changements nécessaires à faire pour passer d'une permutation à l'autre selon l'algorithme du tri à bulle. La distance entre deux permutations de n éléments varie sur $[0; \frac{n(n-1)}{2}]$. Il est intéressant de noter qu'elle peut être calculée en temps log-linéaire.

1.2.1.2 Nombre minimum de transpositions

La distance de transposition [DG77] compte entre deux permutations le nombre minimal de transpositions entre deux éléments nécessaires pour passer d'une permutation à l'autre. On remarque que c'est une métrique La distance entre deux permutations de $\mathbb{S}_n$ varie sur $[0; n-1]$.

1.2.1.3 Distance de Spearman

Cette métrique [DG77] somme la valeur absolue des différences de rangs pour chaque élément des deux permutations π et σ.

Définition 1.6. La distance de Spearman, noté ici F, est définie entre deux permutations de de $\mathbb{S}_n$ de la façon suivante :

$$F(\pi, \sigma) = \sum_{i=1}^{n} |\pi[i] - \sigma[i]| \tag{1.2}$$

La distance entre deux permutations de n éléments varie sur $[0; \frac{n^2}{2}]$.

La distance de Kendall-τ et la distance de Spearman sont équivalentes [DG77] au sens où elles sont liées par un facteur constant :

$$D(\pi, \sigma) \leq F(\pi, \sigma) \leq 2D(\pi, \sigma).$$

1.2.1.4 Norme L_2

La mesure introduite par [DG77] somme le carré des différences entre les rangs des éléments de deux permutations π et σ. C'est la norme L_2, on la définit de la façon suivante :

$$L_2(\pi, \sigma) = \sum_{i=1}^{n} (\pi[i] - \sigma[i])^2$$

La distance entre deux permutations de n éléments varie [DG77] sur $[0; \frac{n^2(n-1)}{3}]$.

1.2.2 Score de Kemeny et consensus optimal

Le **score de Kemeny** [Kem59] d'une permutation π vis-à-vis d'un ensemble de permutations en entrée $\mathcal{P}$ est défini comme la somme des distances de Kendall-τ (*cf.* Équation 1.1) entre π et les permutations de $\mathcal{P}$.

Définition 1.7. Pour tout ensemble de permutations $\mathcal{P} \subseteq \mathbb{S}_n$ et une permutation $\pi \in \mathbb{S}_n$, le score de Kemeny, noté ici S, est :

$$S(\pi, \mathcal{P}) = \sum_{\sigma \in \mathcal{P}} D(\pi, \sigma)$$

Finalement, trouver un consensus optimal π^* d'un ensemble de permutations $\mathcal{P} \subseteq \mathcal{S}_n$ revient à trouver une permutation π^* qui minimise le score de Kemeny envers $\mathcal{P}$.

Définition 1.8. Un consensus optimal $\pi^* \in \mathcal{S}_n$ d'un ensemble de permutations $\mathcal{P} \subseteq \mathcal{S}_n$ est défini de la façon suivante :

$$\forall \pi \in \mathcal{S}_n : S(\pi^*, \mathcal{P}) \leq S(\pi, \mathcal{P}).$$

Il faut noter que l'unicité du consensus optimal n'est pas garantie. Un consensus optimal π^* est aussi appelé *Permutation optimale de Kemeny* (*optimal Kemeny ranking*) [Ail10 ; AM12 ; BK09 ; BBN13], *agrégation optimale* (*optimal agregation*) [Dwo+01], *solution optimale* (*optimal solution*) [ACN08 ; JKS14 ; Mei+12 ; SZ09] ou encore *médiane* (*median ranking*) [CBDH11].

Exemple 1.2. Considérons un ensemble de permutations $\mathcal{P} \subseteq \mathcal{S}_4$. Deux permutations π_1^* et π_2^*, ou consensus optimaux, minimisent le score de Kemeny :

$$\mathcal{P} = \begin{pmatrix} \pi_1 = [A, D, B, C] \\ \pi_2 = [A, C, B, D] \\ \pi_3 = [D, A, C, B] \\ \pi_4 = [D, C, A, B] \end{pmatrix} \qquad \begin{matrix} \pi_1^* = [A, D, C, B] \\ \pi_2^* = [D, A, C, B] \end{matrix}$$

Le score de Kemeny de π_1^* est constitué des désaccords A-C avec π_4, A-D avec π_3 et π_4, D-C avec π_2, D-B avec π_2, et C-B avec π_1 donc $S(\pi_1^*, \mathcal{P}) = 1_{A-C} + 2_{A-D} + 1_{D-C} + 1_{D-B} + 1_{C-B} = 6$. Le score de Kemeny de π_2^* est similaire, et varie

avec les deux désaccords relatifs au couple A-D. Les scores de Kemeny de π_1^* et π_2^* sont alors égaux :

$$S(\pi_1^*, \mathcal{P}) = S(\pi_2^*, \mathcal{P}) = 6$$

1.2.3 Complexité du problème

Lorsque l'on considère la distance Kendall-τ, le problème d'agrégation de classements est NP-Difficile lorsque le nombre de permutations considérées est pair et supérieur ou égal 4 [BBD09 ; Dwo+01]. Aucun résultat n'a été à ce jour publiée quant à la complexité du problème dans le cas où le nombre de permutations est impair.

1.3 Prise en compte des égalités

On considère maintenant le problème d'agrégation de classements dans le cas où des éléments peuvent être considérés comme ex-æquo. Un classement avec égalités n'est alors non pas une suite d'éléments mais une suite de groupes d'éléments.

Définition 1.9. Un **groupe** d'éléments $\mathcal{G}$ est un ensemble d'éléments qui, au sein d'un classement, sont à égalité. Un groupe est nécessairement non-vide.

Définition 1.10. Un **classement avec égalités** r sur $[n]$ est une liste ordonnée de groupes d'éléments $r = [\mathcal{G}_1, \ldots, \mathcal{G}_k]$ telle que ces groupes forment une partition de $[n]$. L'ensemble des classements avec égalités sur $[n]$ forme l'ensemble $\mathcal{R}_n$.

Un classement avec égalités ne contient pas nécessairement d'égalité. Une permutation est alors un cas particulier d'un classement où chaque groupe est un singleton.

Définition 1.11. Soit $r = [\mathcal{G}_1, \ldots, \mathcal{G}_k]$ un classement avec égalités, et soit $\mathcal{G}_l$ un groupe d'éléments de r contenant l'élément x. La position de x dans r est notée $r[x]$ et est définie de la façon suivante :

$$r[x] = 1 + |\bigcup_{i \in [1;l-1]} \mathcal{G}_i|$$

En d'autres termes la position d'un élément est le nombre d'éléments le précédant plus un. On remarque que cette définition est cohérente avec la définition de la position d'un élément dans une permutation.

Propriété 1.1. Par souci de lisibilité, on considérera qu'un **classement** est un classement avec égalités (*cf.* Définition 1.10), et une **permutation** une suite d'éléments sans égalité (*cf.* Définition 1.1)

Un classement avec égalités, ou classement, comme formulé en définition 1.10 est équivalent à un **ordre strict de groupe** (*bucket order*) dans [Fag+06].

Pour plus de lisibilité les exemples traitant de classement, comme les exemples avec des permutations, seront construit avec des lettres.

Exemple 1.3. Soit un classement $r = [\{A\}, \{D\}, \{B, C\}]$, l'élément A est au rang 1 ($r[A] = 1$), suivi de l'élément D au rang 2, le classement est ensuite terminé au rang 3 par les éléments B et C qui sont à égalité ($r[B] = r[C] = 3$).

Définition 1.12. Un **jeu de données** $\mathcal{R}$ sur $[n]$ est un sous-ensemble de $\mathcal{R}_n$.

1.3.1 Mesures de dissimilarité

Pour comparer deux classements avec égalités, deux approches complémentaires ont été suivies, soit en adaptant les mesures existantes [CBDH11 ; Fag+06], soit en proposant de nouvelles mesures [Fag+06].

1.3.1.1 Distance de Kendall-τ généralisée

La généralisation de la distance de Kendall-τ [CBDH11 ; Fag+06] compte les désaccords de paires en distinguant les désaccords d'ordre où il y a une inversion stricte, des désaccords d'égalité où les deux éléments sont dans un même groupe dans un seul des deux classements. Dans le cas d'un désaccord d'égalité une pondération est appliquée, on compte $p \in]0; 1]$ pour chacun de ces désaccords.

Définition 1.13. La distance de Kendall-τ généralisée entre deux classements $r, s \in \mathcal{R}_n$ est notée $G^{(p)}(r, s)$ avec $p \in]0; 1]$ et s'écrit de la façon suivante :

$$\begin{aligned} G^{(p)}(r,s) = &|\{(i,j) : i < j \wedge (r[i] < r[j] \wedge s[i] > s[j] \vee r[i] > r[j] \wedge s[i] < s[j])\}| \\ &+ p * |\{(i,j) : i < j \wedge (r[i] \neq r[j] \wedge s[i] = s[j] \vee r[i] = r[j] \wedge s[i] \neq s[j])\}| \end{aligned}$$

Par abus de langage cette mesure est nommée distance pour $p \in]0;1]$. Lorsque $p \in]0;0.5[$, l'inégalité triangulaire n'est pas respectée et la mesure n'est alors qu'une semi-métrique (*cf.* Définition 1.4) ; pour $p \geq 0.5$ la mesure est une distance [Fag+06]. La distance de Kendall-τ est calculable en temps log-linéaire dans sa version généralisée, et une implémentation est proposée en Appendice B.

Exemple 1.4. Considérons r et s deux classements de $\mathcal{R}_5$.

$$r = [\{E\}, \{C, D\}, \{A, B\}] \qquad s = [\{A, B, D\}, \{E\}, \{C\}]$$

La distance entre ces deux classements au sens de la distance de Kendall-τ généralisée est :

$$G^{(p)}(r, s) = 5 + 3p;$$

1.3.1.2 Distance de Spearman

La distance de Spearman telle que formulée en Définition 1.6 est compatible avec les classements avec égalités [Fag+06], et ne nécessite pas d'adaptation.

Exemple 1.5. En reprenant r et s de l'Exemple 1.4, on a :$F(r, s) = 15$

1.3.1.3 Utilisation de la distance de Hausdorff

La métrique de Hausdorff [Hau27] permet de comparer deux ensembles en utilisant une distance définie pour chaque paire d'éléments.

Définition 1.14. Pour tout ensemble E tel que $A \subseteq E$ et $B \subseteq E$, doté d'une métrique quelconque $d(\gamma_1, \gamma_2)$ définie pour tout $\gamma_1, \gamma_2 \in E$, la distance de Hausdorff, noté ici $d_{Haus}(A, B)$ est définie de la façon suivante :

$$d_{Haus}(A, B) = max\left(\max_{\gamma_1 \in A}\left(\min_{\gamma_2 \in B} d(\gamma_1, \gamma_2)\right), \max_{\gamma_2 \in B}\left(\min_{\gamma_1 \in A} d(\gamma_1, \gamma_2)\right)\right)$$

La métrique a été utilisée avec la distance de Kendall-τ non généralisée dans [Cri84 ; Fag+06], et avec la distance de Spearman dans [Fag+06]. Pour ce faire Fagin *et. al.* [Fag+06] définissent un opérateur $*$ permettant de séparer les éléments ex-æquo d'un classement en fonction d'un autre.

Définition 1.15. Soit l'opérateur $\_*\_$ défini sur $\mathcal{R}_n \times \mathcal{R}_n \to \mathcal{R}_n$ tel que :

$$\forall r, s \in \mathcal{R}_n, i, j \in [n] : (s*r)[i] < (s*r)[j] \iff r[i] < r[j] \vee r[i] = r[j] \wedge s[i] < s[j]$$

Exemple 1.6. Soit $r = [\{E\}, \{C, D\}, \{A, B\}]$ et $\rho = [\{A\}, \{B\}, \{C\}, \{D\}, \{E\}]$, on a alors $\rho * r = [\{E\}, \{C\}, \{D\}, \{A\}, \{B\}]$ où les groupes de r ont été fragmentés suivant l'ordre de ρ.

La distance de Hausdorff peut alors être utilisée soit avec la distance de Kendall-τ soit la distance de Spearman.

Définition 1.16. Soit $\rho \in \mathcal{R}_n$ un classement sans égalité, soit $r, s \in \mathcal{R}_n$, soit r^R l'inversion du classement r ($\forall i, j \in [n], r[i] < r[j] \iff r^R[i] > r^R[j]$) et s^R l'inversion du classement s. Les distances de Kendall-τ et Spearman en utilisant la distance de Hausdorff, noté respectivement K_{Haus} et F_{Haus}, sont alors définies de la façon suivante :

$$K_{Haus}(r, s) = max(K(\rho * r^R * s, \rho * s * r), K(\rho * r * s, \rho * s^R * r))$$
$$F_{Haus}(r, s) = max(F(\rho * r^R * s, \rho * s * r), F(\rho * r * s, \rho * s^R * r))$$

En considérant une distance quelconque D et un classement quelconque ρ où il n'y a pas d'égalité. Une première distance est calculée entre s où ses égalités ont été cassées selon l'ordre inverse de r puis de ρ et r où ses égalités ont été cassées selon l'ordre s puis de ρ. Une seconde distance est calculée entre s où ses égalités ont été cassées selon l'ordre r puis de ρ et r où ses égalités ont été cassées selon inverse de l'ordre s puis de ρ. La distance de Hausdorff entre r et s en utilisant D est alors la plus grande de ces deux distances intermédiaires calculées.

Exemple 1.7. Considérons deux classements de $\mathcal{R}_5$. Soit r et s, ainsi que leurs inverse r^R et s^R.

$$r = [\{E\}, \{C, D\}, \{A, B\}] \quad r^R = [\{A, B\}, \{C, D\}, \{E\}]$$
$$s = [\{A, B, D\}, \{E\}, \{C\}] \quad s^R = [\{C\}, \{E\}, \{A, B, D\}]$$

Prenons un classement sans égalité quelconque de $\rho \in \mathcal{R}_5$:

$$\rho = [\{A\}, \{B\}, \{C\}, \{D\}, \{E\}]$$

Mesurons la distance de Hausdorff utilisant la distance de Kendall-τ entre r et s :

$$K_{Haus}(r,s) = max(K(\underbrace{\rho * r^R * s}_{(1)}, \underbrace{\rho * s * r}_{(2)}), K(\underbrace{\rho * r * s}_{(3)}, \underbrace{\rho * s^R * r}_{(4)}))$$

En détaillant les quatre classements intermédiaires calculés :

$$\begin{aligned}
(1)\ \rho * r^R * s &= \rho * [\{A,B\},\{D\},\{E\},\{C\}] = [\{A\},\{B\},\{D\},\{E\},\{C\}] \\
(2)\ \rho * s * r &= \rho * [\{E\},\{D\},\{C\},\{A,B\}] = [\{E\},\{D\},\{C\},\{A\},\{B\}] \\
(3)\ \rho * r * s &= \rho * [\{D\},\{A,B\},\{E\},\{C\}] = [\{D\},\{A\},\{B\},\{E\},\{C\}] \\
(4)\ \rho * s^R * r &= \rho * [\{E\},\{C\},\{D\},\{A,B\}] = [\{E\},\{C\},\{D\},\{A\},\{B\}]
\end{aligned}$$

On a donc

$$K(\underbrace{[\{A\},\{B\},\{D\},\{E\},\{C\}]}_{(1)}, \underbrace{[\{E\},\{D\},\{C\},\{A\},\{B\}]}_{(2)}) = 7$$
$$K(\underbrace{[\{D\},\{A\},\{B\},\{E\},\{C\}]}_{(3)}, \underbrace{[\{E\},\{C\},\{D\},\{A\},\{B\}]}_{(4)}) = 6$$

Finalement

$$K_{Haus}(r,s) = \max(7,6) = 7$$

On remarque que si on avait prit un quelconque autre ρ la distance aurait été la même, en effet les éléments $\{A,B\}$ aurait été fragmenté d'une façon ou d'une autre mais n'aurait de toute façon pas ajoutés de désaccord à la distance globale.

1.3.1.4 Liens entres les distances

Définissons dans un premier temps la notion d'équivalence entre deux distances.

Définition 1.17. D_1 et D_2, deux distances de $A \times A \to B$, sont équivalentes $\iff \exists \alpha, \beta \in \mathbb{R}^+$ tq $\forall \sigma_1, \sigma_2 \in A\ \alpha D_1(\sigma_1,\sigma_2) \leq D_2(\sigma_1,\sigma_2) \leq \beta D_2(\sigma_1,\sigma_2)$

Dans [Fag+06], les auteurs ont prouvé que les distances de Kendall-τ généralisée, la distance de Spearman, et les distances de Hausdorff sont équivalente entre elles. Plus précisément ils ont prouvé que la distance de Kendall-τ généralisée avec

le paramètre $p = \frac{1}{2}$ était équivalente à elle-même pour toute valeur de p positive :

$$\forall p > 0, \exists \alpha, \beta \in \mathbb{R}^+ \text{ tq } \alpha G^{(\frac{1}{2})}(\pi, \sigma) \leq G^{(p)}(\pi, \sigma) \leq \beta G^{(\frac{1}{2})}(\pi, \sigma). \tag{1.3}$$

Ensuite ils ont prouvé que les distances était équivalente entre elle ;

$$G^{(\frac{1}{2})}(\pi, \sigma) \leq F(\pi, \sigma) \leq 2G^{(\frac{1}{2})}(\pi, \sigma). \tag{1.4}$$

$$K_{Haus}(\pi, \sigma) \leq F_{Haus}(\pi, \sigma) \leq 2K_{Haus}(\pi, \sigma). \tag{1.5}$$

$$G^{(\frac{1}{2})}(\pi, \sigma) \leq K_{Haus}(\pi, \sigma) \leq 2G^{(\frac{1}{2})}(\pi, \sigma). \tag{1.6}$$

Utiliser une de ces distance équivaut donc, à un facteur 4 près, à utiliser chacune des autres distances.

1.3.2 Score de Kemeny généralisé

La généralisation du score de Kemeny se fait en calculant la somme des distances de Kendall-τ généralisées :

Définition 1.18. Pour tout classement avec égalités $r \in \mathcal{R}_n$ et un ensemble de classements $\mathcal{R} \subseteq \mathcal{R}_n$, le score de Kemeny de r par rapport à $\mathcal{R}$ est défini en fonction de $p \in]0; 1]$:

$$K^{(p)}(r, \mathcal{R}) = \sum_{s \in \mathcal{R}} G^{(p)}(r, s)$$

Suivant la définition précédente, le score entre un ensemble classement $\mathcal{R}$ et un classement r avec $p = 1$ s'écrit $K^{(1)}(r, \mathcal{R})$. Par souci de lisibilité, lorsque $p = 1$, nous écrivons le score $K(r, \mathcal{R})$.

Trouver le consensus optimal r^* d'un ensemble de classements $\mathcal{R} \subseteq \mathcal{R}_n$ revient à trouver un classement r^* minimisant le score de Kemeny généralisé.

Définition 1.19. Le consensus optimal $r^* \in \mathcal{R}_n$ d'un ensemble de classements avec égalités $\mathcal{R} \subseteq \mathcal{R}_n$ est défini de la façon suivante :

$$\forall r \in \mathcal{R}_n : K^{(p)}(r^*, \mathcal{R}) \leq K^{(p)}(r, \mathcal{R})$$

Exemple 1.8. Considérons un ensemble de classements $\mathcal{R} \subseteq \mathcal{R}_4$ et son consensus optimal r^* au sens du score de Kemeny généralisé avec $p = 1$:

$$\mathcal{P} = \begin{pmatrix} r_1 = [\{A\}, \{D, B\}, \{C\}] \\ r_2 = [\{A\}, \{C, B\}, \{D\}] \\ r_3 = [\{D, A\}, \{C, B\}] \\ r_4 = [\{D\}, \{C\}, \{A\}, \{B\}] \end{pmatrix} \qquad r_1^* = [\{A\}, \{D\}, \{B, C\}]$$

Le score de Kemeny généralisé de r^* est constitué des inversions strictes suivantes : A-C avec r_4, A-D avec r_4, B-D avec r_2, C-D avec r_2, et des désaccords d'égalités suivant : A-D avec r_3, B-C avec r_1 et r_4, B-D avec r_1. On a donc $K(r^*, \mathcal{R}) = 4 + 4 * p = 8$.

1.3.3 Discussion

Il est intéressant de noter que parmi les distances présentées, la distance de Kendall-τ et sa généralisation sont, à notre connaissance, constamment utilisées dans le cas des problèmes d'agrégation de classements. Cette popularité peut être justifiée par quatre points : (1) l'antériorité de la formalisation du *problème d'agrégation de rang de Kemeny*[Kem59] qui utilise la distance de Kendall-τ, (2) sa simplicité (compter les paires en désaccord entre les deux classements), (3) une équivalence avec plusieurs autres distances à un facteur près, et finalement par (4) une implémentation qui permet de calculer le nombre de désaccords entre deux classements en un temps log-linéaire.

Par la suite nous considérons le problème de l'agrégation de classements avec égalités uniquement au regard de la distance de Kendall-τ généralisée.

1.4 Prise en compte des éléments manquants

Dans les cas réels d'utilisation, les classements des jeux de données ne contiennent pas nécessairement tous les mêmes éléments [BBN13 ; CBDH11 ; Dwo+01 ; SZ09]. Or, les classements des jeux de données jusque-là considéré contienne toujours les mêmes éléments. Afin de distinguer ces deux cas, on appelle les classements et permutations contenant les mêmes éléments des classements *complets*, des permutations *complètes* et des jeux de données *complets*. Par opposition lorsqu'ils ne contiennent pas nécessairement les mêmes éléments, on dira qu'ils sont *incomplets*. Nous formalisons donc ci-après l'absence d'élément en introduisant les concepts de permutations, classements et jeux de données *incomplets*.

Définition 1.20. Une permutation incomplète π est une bijection d'un sous-ensemble de $[n] = \{1, 2, ..., n\}$ sur lui-même, en d'autres termes c'est un ordre total strict d'un sous-ensemble de $[n]$.

L'ensemble des permutations incomplètes de $[n]$ est noté $\mathbb{S}_n^I$. On remarque que l'ensemble des permutations complètes est un sous-ensemble des permutations incomplète : $\mathbb{S}_n \subset \mathbb{S}_n^I$.

Définition 1.21. Un classement (avec égalités) incomplet sur $[n] = \{1, 2, ..., n\}$ est une liste de sous-ensembles $r = [\mathcal{G}_1, \dots \mathcal{G}_k]$ telle que $\mathcal{G}_1, \dots, \mathcal{G}_k$ forment une partition d'un sous-ensemble de $[n]$. On dit que x est ordonné avant y si et seulement s'il existe i, j avec $i < j$ et $x \in \mathcal{G}_i$ et $y \in \mathcal{G}_j$.

L'ensemble des classements incomplets de $[n]$ est noté $\mathcal{R}_n^I$. De plus, $\mathcal{R}_n \subset \mathcal{R}_n^I$.

Exemple 1.9. Soit $\mathcal{R} \subseteq \mathcal{R}_5^I$ un jeu de données composé de 4 classements incomplets sur $\{A, B, C, D, E\}$. L'élément D dans le classement r_1 est au rang 2.

$$\mathcal{R} = \begin{pmatrix} r_1 = [\{A\}, \{D, E\}] \\ r_2 = [\{A\}, \{E\}, \{D\}] \\ r_3 = [\{B\}, \{A, D\}, \{C\}] \\ r_4 = [\{A\}, \{D\}] \end{pmatrix}$$

Par soucie de clarté, un *classement incomplet* sera explicitement dénoté **classement incomplet**, alors qu'un *classement complet* sera appelé **classement**. Il en ira de même pour les jeux de données et permutations.

1.5 Notation et utilisation

1.5.1 Éléments présent dans un classement

Lorsqu'un classement est incomplet, la position d'un élément dans ce classement n'est pas nécessairement définit. On considère alors une fonction $elem(r)$ qui permet de retourner l'ensemble des éléments présents dans r et qui ont donc une position.

1.5.2 Éléments avant et après un élément

Par définition (*cf.* Définition 1.11) le nombre d'éléments placé avant un élément x au sein d'un classement r est noté $r[x] - 1$.

On définit une fonction $après(r, x)$ qui retourne le nombre d'éléments situés après x dans le classement r, qu'il soit ou non complet :

Définition 1.22. Soit $r = [\mathcal{G}_1, \ldots, \mathcal{G}_k]$ un classement avec égalités, et soit $\mathcal{G}_l$ un groupe d'éléments de r contenant l'élément x. Le nombre d'éléments après x est défini de la façon suivante :

$$après(r, x) = |\bigcup_{i \in [l+1;k]} \mathcal{G}_i|$$

Chapitre 2

État de l'art

Sommaire

2.1 Introduction

L'agrégation de classements trouve de nombreuses applications, des élections [Bor81] au filtrage de résultats de recherche sur le web pour réduire le spa m [Dwo+01] en passant par le calcul de consensus entre experts [Sta+14]. Dans ce chapitre nous présentons une vue d'ensemble des approches s'inscrivant dans le cadre de l'agrégation de classements, tant pour la production d'un consensus à

partir d'un ensemble de classements, que pour préparer les données réelles à un tel calcul. Nous présenterons ensuite les résultats précédemment obtenus en comparant divers algorithmes calculant des consensus.

Plus précisément (1) nous présentons les algorithmes calculant des consensus en les séparant en fonction de l'information de base exploitée, à savoir les désaccords entre paires d'éléments (avec et sans prise en compte des égalités) ou encore la position des éléments. Dans un second temps (2) nous étudions les diverses approches dites de *normalisation* utilisées pour faire le lien entre les jeux de données réels et les contraintes qu'impose le problème de l'agrégation de classements tel qu'énoncé au Chapitre 1. Enfin (3) nous synthétisons les résultats publiés dans la littérature évaluant la qualité des consensus produits par les algorithmes et le temps requis pour accomplir cette tâche. Finalement (4) nous concluons avec les perspectives de travail qu'ouvre cet état de l'art.

2.2 Classification des approches

Dans cette section, nous présentons les différents algorithmes qui ont été conçus et sont disponibles dans la littérature pour agréger des classements. La classification des algorithmes se fait en fonction de leur capacité à prendre en compte, ou non, les égalités entre éléments dans les classements, et à y associer un coût, une des problématiques majeures associées à l'utilisation de données réelles et mises en avant dans l'introduction. Pour associer un coût lorsque l'on crée ou rompt une égalité entre deux éléments, la distance de Kendall-τ généralisée est nécessaire. Nous présenterons donc dans un premier temps les algorithmes l'utilisant (Section 2.2.1, [G] en Table 2.1). Dans un second temps nous présenterons les algorithmes utilisant la distance de Kendall-τ classique (Section 2.2.2, [K] en Table 2.1). Finalement nous présenterons les algorithmes *positionnels*. Ces derniers ne se concentrent pas sur les paires d'éléments et les désaccords associés, mais sur la position des éléments dans les classements (Section 2.2.3, [P] en Table 2.1). Les algorithmes seront accompagnés d'un exemple de résultat calculé par rapport au jeu de données fourni en Exemple 2.1.

2.2.1 Algorithmes basés sur la distance de Kendall-τ généralisée

Nous présentons ici les deux seuls algorithmes publiés conçus pour gérer le coût de créer/rompre des égalités et donc de produire en sortie des classements avec égalités au sens de la distance de Kendall-τ généralisée.

Ref	Nom	Approx.	Classe d'algorithme
[Ail10]	**Ailon$\frac{3}{2}$**	3/2	[K] Programmation en nombres entiers
[CBDH11]	**BioConsert**	2	[G] Recherche locale
[Bor81] [CFR06]	**BordaCount**	5	[P] Tri par score
[CK96]	Chanas	no	[K] Recherche locale
[CW09]	ChanasBoth	no	[K] Recherche locale
[AM12]	BnB	exact	[K] Branch & Bound
[Cop51]	**CopelandMethod**	no	[P] Tri par score
[Fag+04]	**FaginDyn**	4	[G] Programmation Dynamique
[AM12] [BBN13] [CDK06] [SZ09]	**ILP**	exacte	[K] Programmation en nombre entier
[ACN08]	**KwikSort**	$\frac{11}{7}$	[K] Diviser pour régner
[Dwo+01]	MC4	no	[P] Hybride
[FKS03a]	**MEDRank**	no	[P] Top-k
[ACN08]	**Pick-a-Perm**	2	[K] Naïf
[Ail10]	**RepeatChoice**	2	[K] Tri par relation d'ordre

Table 2.1 – Algorithmes et leurs catégories, "Approx." est une abréviation d'approximation, "[P]" indique un algorithme positionnel, "[K]" un algorithme basé sur la distance de Kendall-τ et "[G]" sur la distance de Kendall-τ généralisée. Les algorithmes en gras ont été ré-implémentés par nos soins et seront évalués au chapitre 4.

Exemple 2.1. Soit $\mathcal{R}$ un jeu de données contenant $m = 4$ classements sur $n = 5$ éléments, et son consensus r^* qui est optimal. Ce jeu de données est utilisé dans cette section pour illustrer les différences entre les algorithmes, nous fournissons donc pour chaque algorithme un exemple de résultat qu'il peut obtenir. :

$$\mathcal{R} \begin{cases} [\{A,B,C\},\{E\},\{D\}] \\ [\{B,D\},\{E\},\{A,C\}] \\ [\{C,D\},\{B\},\{A,E\}] \\ [\{A\},\{E\},\{C,D\},\{B\}] \end{cases}$$

$$r^* = [\{C,D\},\{B\},\{A\},\{E\}] \quad K(\mathcal{R},r^*) = 20$$

FaginDyn [Fag+04] est une approche reposant sur de la programmation dynamique. Deux variantes ont été proposées par [CBDH11], la première (FaginLarge) favorisant la production de consensus avec de grands groupes d'éléments à égalités, la seconde (FaginSmall) tendant à produire des consensus avec peu d'égalités. La construction d'une solution par programmation dynamique se base sur la construction préalable d'une sous-solution que l'on suppose optimale. Lorsque plusieurs sous-solutions sont éligibles, les variantes FaginSmall et FaginLarge se distinguent en prenant, respectivement, la dernière ou la première des meilleures sous-solutions rencontrées.

La complexité de FaginDyn en temps est en $\mathcal{O}(nm + n^2)$ [1], et sa complexité mémoire est dominée par la représentation matricielle sous-jacente en $\mathcal{O}(n^2)$.

Exemple 2.2. Consensus calculé par FaginSmall (r_s) et FaginLarge (r_l) :

$$\begin{array}{ll} r_s = [\{B\},\{A\},\{C\},\{E\},\{D\}] & K(\mathcal{R}, r_b) = 23 \\ r_l = [\{B,A,C,E\},\{D\}] & K(\mathcal{R}, r_b) = 29 \end{array}$$

BioConsert [CBDH11] est la seconde approche capable de prendre en compte le coût de créer/rompre des égalités. BioConsert suit une approche de **recherche locale** (*local search*) gloutonne. A partir d'un classement de départ, il applique des opérations d'édition pour modifier la solution pour peu que cela réduise le coût de la solution actuelle. Le ou les classements de départ peuvent être choisis comme étant les classements en entrée eux-mêmes [CBDH11], mais aussi le(s) résultat(s) d'autre(s) algorithme(s) [Bra+14]. Les deux opérations d'édition de BioConsert sont le retrait d'un élément de son groupe courant pour le placer dans un groupe (i) nouveau ou (ii) pré-existant (*cf.* Figure 2.1).

L'implémentation de BioConsert mise en ligne par les auteurs de [CBDH11] a une complexité en mémoire en $\mathcal{O}(n^2)$.

Exemple 2.3. Consensus (optimal) calculé par BioConsert :

$$r_b = [\{C,D\},\{B\},\{A\},\{E\}] \qquad K(\mathcal{R}, r_b) = 20$$

2.2.2 Algorithmes basés sur la distance de Kendall-τ

On se concentre maintenant sur les algorithmes utilisant la distance de Kendall-τ. Même si elle est faite pour des permutations, l'énonciation de la distance de

1. Dans un jeu de données il y a n éléments et m classement

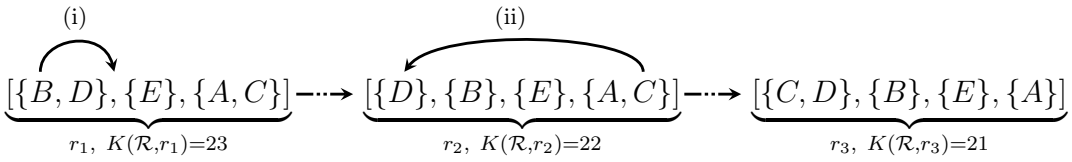

FIGURE 2.1 – Opérations d'édition dans BIOCONSERT. En se basant sur le jeu de données $\mathcal{R}$ présenté en Exemple 2.1, le mouvement (i) déplace B dans un nouveau groupe entre D et E pour produire r_2 en réduisant le coût de 1. Le mouvement (ii) déplace C pour le mettre dans le même groupe que D et produire r_3 et réduire de nouveau le coût de 1. Á noter qu'un dernier mouvement pour mettre A entre B et E permet d'obtenir le consensus optimal.

Kendall-τ permet de l'utiliser avec des classements avec égalités comme dans [SZ09 ; AM12]. Cela revient à utiliser la distance de Kendall-τ généralisée avec le paramètre $p = 0$ (*cf.* Définition 1.13 p.17), cependant utiliser $p = 0$ implique que l'on doive la qualifier non pas de distance mais de métrique [Fag+06]. On remarque en effet qu'elle ne respecte (entre autre) pas l'identité :

$$\text{Soit } r_1 = [\{B\},\{A\}], r_2 = [\{A,B\}],\ K^{(0)}(r_1,r_2) = 0 \text{ or } r_1 \not\equiv r_2$$

En utilisant l'Exemple 2.12, la conséquence du fait qu'il n'y ait pas de coût à créer et rompre des égalités, est que le consensus optimal est alors $[\{A,B,C,D,E\}]$. Afin d'éviter ce résultat absurde, les algorithmes utilisant la mesure de Kendall-τ avec des classements avec égalités en entrée produisent nécessairement en sortie des classements sans égalité, des permutations.

Détaillons ci après les algorithmes produisant des permutations avec la distance de Kendall-τ.

2.2.2.1 Programmation linéaire

Le problème de l'agrégation de classements a naturellement été traduit dans le cas d'agrégation de permutations en problème de programmation en nombres entiers (PNE) [AM12 ; CDK06 ; SZ09]. Un problème de PNE, s'exprime avec plusieurs variables entières, des contraintes qui sont des combinaisons des dites variables, et une fonction objectif (une combinaison linéaire des variables) à minimiser. Dans [CDK06], les auteurs proposent une formulation en PNE du problème de l'agrégation de permutations que nous reprenons ci-après en simplifiant la notation. Dans la suite, $\mathcal{P}$ dénote un ensemble de m permutations ordonnant les éléments de $[n]$.

Variable. Pour chaque paire d'éléments (a, b), on définit une variable $x_{a<b}$ afin d'exprimer que dans le consensus optimal a est avant b, la variable prenant comme valeur 0 ou 1.

$$\forall a, b \in [n] : x_{a<b} \in \{0, 1\}$$

On définit $w_{a<b}$ afin de compter le nombre de classements dans lesquels a est effectivement avant b. On utilise pour ce faire la convention d'Iverson [Gra94, p24] qui permet d'encapsuler une expression booléenne entre deux crochets, et de considérer que le résultat vaut un quand l'expression est vraie ($[1 > 0] = 1$), et zéro sinon ($[1 + 1 = 3] = 0$).

$$\forall a, b \in [n] : w_{a<b} = \sum_{\pi \in \mathcal{P}} [\pi[a] < \pi[b]]$$

Fonction objectif. L'objectif du PNE est d'exprimer la distance de Kendall-τ en utilisant les variables précédemment introduites. Il restera alors à la charge de l'outil résolvant le PNE (CPLEX, LPSolve, GLPK, ...) de trouver une allocation des variables minimisant la fonction objectif, et donc la distance de Kendall-τ :

$$\sum_{\substack{a,b \in [n] \\ a<b}} (w_{b<a} * x_{a<b} + w_{a<b} * x_{b<a})$$

Contraintes. Elles doivent permettre de produire une solution qui soit une permutation. Pour toute paire d'éléments (a, b), l'élément a doit être soit avant soit après l'élément b, on a donc :

$$\forall a, b \in [n] : x_{a<b} + x_{b<a} = 1 \tag{2.1}$$

Afin de conserver la transitivité dans une permutation, à savoir que si a est avant b et b avant c alors a est nécessairement avant c, on a finalement :

$$\forall a, b, c \in [n] : x_{a<c} - x_{a<b} - x_{b<c} \geq -1 \tag{2.2}$$

Utilisation. Ce programme a aussi été utilisé avec des classements avec égalités [AM12 ; SZ09], dans ce cas les solutions retournées peuvent être non-optimales comme le montre l'exemple 2.4.

Exemple 2.4. Consensus calculé par programmation en nombres entiers :

$$r_b = [\{D\}, \{C\}, \{B\}, \{A\}, \{E\}] \qquad K(\mathcal{R}, r_b) = 21$$

Ailon [Ail10] a proposé une $\frac{3}{2}$-approximation (on l'appellera ici AILON$\frac{3}{2}$) basée sur une relaxation du PNE en programme linéaire en nombres réels. Une fois résolu, la reconstruction du consensus se fait en arrondissant à l'entier le plus proche les variables réelles.

Exemple 2.5. Consensus calculé par AILON$\frac{3}{2}$:

$$r_b = [\{C\}, \{D\}, \{B\}, \{A\}, \{E\}] \qquad K(\mathcal{R}, r_b) = 21$$

2.2.2.2 Recherche arborescente

KWIKSORT [ACN08] est un algorithme d'approximation basé sur une approche *diviser pour régner* (*divide-and-conquer*). Lorsque l'on choisit la meilleure solution entre celle de KWIKSORT et celle de PICK-A-PERM (un algorithme naïf qui sera introduit en Exemple 2.2.2.5), on est alors assuré d'avoir un ratio d'approximation de $\frac{11}{7}$ [ACN08].

En considérant un ensemble d'éléments, KWIKSORT choisit aléatoirement un élément pivot, et assigne les autres éléments dans deux groupes placés *avant* et *après* le pivot de telle façon qu'ils minimisent leurs désaccords avec l'élément pivot. L'algorithme se lance ensuite récursivement sur les deux groupes *avant* et *après*.

La consommation en mémoire et en temps de cet algorithme est en $\mathcal{O}(n\ log(n))$, où n est le nombre d'éléments.

Une version déterministe a été présentée [VZW09] et a un ratio d'approximation de $\frac{8}{5}$. Dans cette variante, le choix du pivot n'est plus aléatoire mais vise à minimiser les désaccords vis-à-vis des éléments qui sont placés dans les groupes *avant* et *après*.

Exemple 2.6. Deux consensus calculés par KWIKSORT :

$$r_k = [\{D\}, \{A\}, \{C\}, \{B\}, \{E\}] \quad K(\mathcal{R}, r_b) = 22$$
$$r'_k = [\{C\}, \{A\}, \{E\}, \{D\}, \{B\}] \quad K(\mathcal{R}, r_b) = 24$$

Dans [AM12] une approche en *Séparation et évaluation* (*branch-and-bound*), que l'on nommera B&B, explore un arbre où chaque feuille à une profondeur j représente une partie d'une solution sur les éléments 1 à j. L'exploration de l'arbre se fait en étudiant la feuille qui a le moins de désaccords par rapport aux permutations fournies en entrée. Lorsque l'on ne limite pas la mémoire utilisée, la solution retournée par cet algorithme est optimale.

Une variante limitant le nombre de feuilles développées par une approche dite de *recherche en faisceau* (*beam search*) est aussi proposée par les auteurs. L'idée principale est qu'à chaque étape de l'exploration où de nouveaux nœuds sont découverts, on ne conserve qu'un nombre limité de nœuds ayant un coût heuristique faible.

2.2.2.3 Pré-traitement en recherche arborescente et résolution en PNE

Alors que KwikSort choisit un pivot aléatoirement pour diviser en deux ensembles les autres éléments, et se relance récursivement, les auteurs de [BBN13] choisissent un pivot bien particulier : un élément propre (*non-dirty element*).

Elément propre : Pour [BBN13] un élément est propre s'il apparaît vis-à-vis de chaque autre élément avant (ou après) dans au moins $\frac{3}{4}$ des classements dans un jeu de données $\mathcal{P}$. En réutilisant les notations introduites en Section 2.2.2.1, un élément $i \in [n]$ est dit propre dans $\mathcal{P}$, noté $propre(i, \mathcal{P})$ si et seulement si :

$$propre(i, \mathcal{P}) \iff \forall j \in [n] \backslash \{i\}, \max(w_{i<j}, w_{j<i}) \geq \frac{3m}{4}$$

Les auteurs ont prouvé que la relation d'ordre majoritaire entre un élément propre et un autre élément (donc d'au moins $\frac{3}{4}$ des classements) était nécessairement conservée dans le consensus optimal.

Les éléments sont alors récursivement divisés tant qu'il existe des éléments propres. Lorsqu'un ensemble ne contient pas d'élément propre, un consensus optimal est calculé entre les éléments de cet ensemble, en PNE (*cf.* Section 2.2.2.1). Le consensus optimal global est alors la concaténation des consensus optimaux de chaque sous-problème du jeu de données. Une exécution du prétraitement est présenté en Figure 2.2.

2.2.2.4 Recherche locale

Alors que les approches par recherche arborescente précédemment présentées construisent progressivement la solution, les approches par recherche locale considèrent une solution déjà construite et la modifient par diverses méthodes jusqu'à ce qu'on ne puisse plus améliorer la solution.

Chanas [CK96] puis ChanasBoth [CW09 ; SZ09] sont des approches gloutonnes de recherche locale basées sur une opération d'édition qui déplace un élément si cela réduit le nombre de désaccords. Dans Chanas, les éléments d'un classement sont successivement observés du premier au dernier, et un élément ne peut être déplacé qu'en direction du premier, si cela réduit le nombre de désaccords du classement avec le jeu de données. Une fois la permutation parcourue, elle est

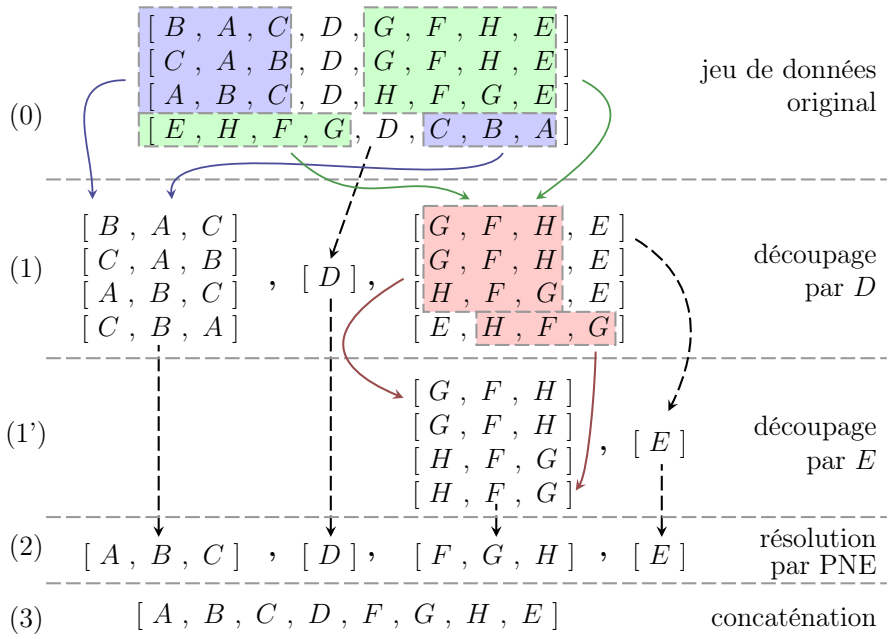

FIGURE 2.2 – L'élément D est vis-à-vis de chaque autre élément soit dans $\frac{3}{4}$ des classements après ces éléments (A, B, C), soit dans $\frac{3}{4}$ des classements avant (E, F, G, H), et peut donc permettre de découper (0) le jeu de données. De même l'élément E est après l'ensemble des éléments dans $\frac{3}{4}$ des classements, et permet aussi de découper (1'). Les découpes (1) et (1') en suivant la règle de majorité des $\frac{3}{4}$ de [BBN13] ont permis de créer deux jeux de données à résoudre indépendamment. Ces derniers sont résolus (2) par PNE (seul un des nombreux consensus est affiché ici), finalement les sous-consensus optimaux sont concaténés (3) pour former un consensus optimal.

inversée (le premier devient le dernier, le second devient l'avant-dernier, et ainsi de suite), et la phase de déplacement est relancée. Dans CHANASBOTH, les éléments peuvent être déplacés dans les deux sens. Une fois la permutation parcourue, elle est ici aussi inversée.

2.2.2.5 Algorithme naïf et son raffinement

PICK-A-PERM est un algorithme naïf [ACN08] qui retourne aléatoirement une des permutations fournies en entrée. Cet algorithme est une 2-approximation [ACN08] lorsque l'on ne considère que les permutations. Une version déterministe a été utilisée par [SZ09], et retourne le classement qui a le moins de désaccords avec les autres. Cet algorithme peut être utilisé avec des classements, mais dans ce cas la borne d'approximation n'est plus garantie.

La version non-déterministe a une complexité en temps constant. La version deterministe a une complexité qui est dominée par la complexité du calcul de m distances.

Exemple 2.7. Consensus calculé par Pick-a-Perm :

$$r_b = [\{A\}, \{E\}, \{C, D\}, \{B\}] \qquad K(\mathcal{R}, r_b) = 23$$

RepeatChoice [Ail10] est une 2-approximation qui est un raffinement de Pick-a-Perm, et produit des permutations. En partant d'un des classements choisis au hasard, les groupes sont fragmentés en suivant l'ordre d'un autre des classements d'entrée aléatoirement choisi, cette fragmentation est répétée jusqu'à ce que l'ensemble des classements en entrée aient été tirés au hasard. S'il reste encore des égalités, les groupes sont aléatoirement fragmentés.

Il est possible de faire une implémentation de cet algorithme avec une complexité en temps de $\mathcal{O}(m \times S(n))$ où $S(n)$ est la complexité de l'algorithme de tri, n le nombre d'éléments, et m le nombre de classements en entrée.

Exemple 2.8. Deux consensus calculés par RepeatChoice :

$$r_r = [\{B\}, \{D\}, \{E\}, \{C\}, \{A\}] \quad K(\mathcal{R}, r_r) = 24$$
$$r_r = [\{B\}, \{C\}, \{A\}, \{E\}, \{D\}] \quad K(\mathcal{R}, r_r) = 23$$

2.2.3 Algorithmes Positionnels

Les approches décrites ici utilisent la position des éléments pour produire un consensus. Elles ont toutes été conçues pour produire des permutations, c'est-à-dire des classements sans égalité.

2.2.3.1 Systèmes de vote

Le système de vote le plus répandu dans les démocraties modernes est une voix qu'un électeur donne à un candidat. Le candidat avec le plus de voix remporte l'élection. Dans son *Mémoire sur les élections au scrutin (1781)*, M de Borda y expose les limitations de ce système dès qu'il y a plus de deux candidats. Il propose un exemple où, entre trois candidats, celui qui est élu est celui qui est pourtant le moins apprécié par les électeurs.

Afin de pallier la faiblesse du système de vote avec une voix unique, M de Borda expose dans son mémoire les prémisses de la production d'un classement

consensuel qui minimiserait les désaccords entre chaque paire de candidats. En effet pour chaque électeur, des préférences sont exprimées entre chaque paire de candidats, puis de ces préférences une permutation par ordre de *mérite* est créée. On rejoint alors le problème de l'agrégation de classements avec une permutation à produire en fonction d'un ensemble de permutations. Dans la **méthode de Borda** (BordaCount), les candidats sont triés par le mérite global qu'ils ont reçu de chaque électeur, le mérite étant dans ce cas équivalent au rang. Les candidats sont alors triés de façon croissante par rapport à la somme des rangs qu'ils ont obtenus. Cet algorithme est aussi appelé *somme des rangs*.

La complexité en temps est en $\mathcal{O}(nm + S(n))$, et la complexité mémoire est dominée par celle de l'algorithme de tri.

Exemple 2.9. Consensus calculé par BordaCount :

$$r_b = [\{C\}, \{A, B, D\}, \{E\}] \qquad K(\mathcal{R}, r_b) = 25$$

La méthode de Copeland (CopelandMethod) [Cop51] fait elle aussi la somme des scores obtenus par chaque candidat dans chaque classement pour ensuite trier les dits candidats. Elle diffère cependant en considérant que le score n'est pas le rang d'un élément dans un classement (c'est-à-dire le nombre d'éléments le précédant), mais le nombre d'éléments le suivant. Comme remarqué par [SZ09], lorsque les classements sont des permutations, les deux méthodes produisent le même résultat.

La complexité en temps est elle aussi en $\mathcal{O}(nm + S(n))$ avec une complexité mémoire linéaire.

Exemple 2.10. Consensus calculé par CopelandMethode :

$$r_b = [\{B, C, D\}, \{A\}, \{E\}] \qquad K(\mathcal{R}, r_b) = 22$$

2.2.3.2 Top-k

Dans les procédures précédentes, une étape de tri est nécessaire, ce qui à pour conséquence que les approches ont une complexité en temps dominée par cette étape finale de tri. *MEDRank* [FKS03b] est un algorithme rapide qui s'inscrit dans la catégorie des approches top-k en permettant de ne retourner que les k premiers éléments, et qui permet surtout d'éviter l'étape de tri. Les classements en entrée sont lus en parallèle, élément par élément, et le consensus est construit au fur et à

mesure. En considérant m classements en entrée, et un paramètre de seuil $h \in]0;1[$, dès qu'un élément a été lu dans $h \times m$ classements, il est ajouté au consensus.

Sa complexité en temps est en $\mathcal{O}(nm)$, et en mémoire en $\mathcal{O}(n)$.

Exemple 2.11. Consensus calculé par MEDRank :

$$r_m = [\{A\}, \{B\}, \{D\}, \{C\}, \{E\}] \qquad K(\mathcal{R}, r_m) = 24$$

2.2.3.3 Autre approche

L'approche MC4 [Dwo+01] (qualifié d'hybride par [SZ09]) représente le problème de l'agrégation de classements avec une chaîne de Markov où les états sont les éléments des classements en entrée. La transition entre deux éléments e_1 et e_2 est créée avec une probabilité de transition égale à $\frac{1}{n}$ s'il y a une majorité des classements préférant e_2 à e_1 (n est le nombre des éléments). Pour produire le consensus, la distribution stationnaire de la chaîne est calculée, et les éléments prennent pour score la probabilité dans la distribution stationnaire de leur état associé. Les éléments sont ensuite triés dans l'ordre croissant.

La complexité de cette méthode est dominée par la complexité de la recherche de la distribution stationnaire.

Nous avons présenté dans cette section différents algorithmes qui ont été conçus pour agréger des classements. Dans la section suivante nous étudions les approches qui permettent aux algorithmes présentés de prendre en comptes les jeux de données incomplets.

2.3 Approches pour travailler avec des jeux de données incomplets

Les jeux de données réels, qu'ils soient faits de classements de données biomédicale [CBDH11], de compétiteurs sportifs [BBN13] ou de page web [Dwo+01], ne contiennent pas nécessairement les mêmes éléments. En effet un compétiteur peut, par exemple, avoir déclaré forfait pour une course et n'est donc classé que dans une partie des classements seulement. On dit alors que le jeu de données est incomplet (*cf.* Définition 1.21 et Exemple 2.12). Le problème de l'agrégation de classements, tel que défini dans le chapitre 1, impose d'avoir des jeux de données où les classements ont les mêmes éléments, c'est-à-dire des jeux de données complets. Deux stratégies permettent alors de faire le lien entre des jeux de données incomplets et

Nom & publications	Jeux de données contenant les même éléments	Utilisés avec des égalités	
EachMovie [CW09]	non	non	
F1 [BBN13]	projeté	non	
BioMedical [CBDH11]	unifié	oui	
GiantSlalom [AM12]	unifié fragmenté	non	
SkiCross/Jumping **[BBN13]**	projeté	non	
WebCommunities [CW09 ; SZ09]	oui	non	
WebSearch [Dwo+01 ; SZ09]	unifié[SZ09 ; AM12]	oui	
[AM12]**[BBN13]**	projeté[BBN13]	non	

TABLE 2.2 – Jeux de données utilisés dans la littérature et utilisation dans les travaux respectifs. Les auteurs ayant rendu disponible leur jeux de données sont indiqués par une référence bibliographique en gras. .

un problème défini pour des jeux de données complets : (i) normaliser le jeu de données pour le rendre complet, (ii) adapter la distance utilisée dans le problème de l'agrégation de classements pour qu'elle gère les jeux de données incomplets.

Dans cette section nous présentons deux approches de normalisation et l'adaptation d'une distance pour qu'elle puisse être utilisée lorsque des éléments sont absents. Nous présentons en Section 2.3.1 la *projection* qui consiste à retirer les éléments absents d'au moins un classement, en Section 2.3.2 l'*unification* qui consiste pour chaque classement à ajouter les éléments manquants à la fin, et finalement en Section 2.3.3 la mesure de Kendall-τ induite qui adapte la distance pour remédier l'absence d'élément. Afin d'illustrer chaque approche, nous les appliquons sur le jeu de données incomplet donné ci-après.

Exemple 2.12. Soit $\mathcal{R} \subseteq \mathcal{R}_5^I$ un jeu de données composé de 4 classements incomplets sur $\{A, B, C, D, E\}$. L'élément D dans le classement r_1 est au rang 2.

$$\mathcal{R} = \begin{pmatrix} r_1 = [\{A\}, \{D\}, \{E\}] \\ r_2 = [\{A\}, \{E\}, \{C\}, \{D\}] \\ r_3 = [\{B\}, \{D\}, \{A\}, \{C\}] \\ r_4 = [\{A\}, \{D\}] \end{pmatrix}$$

Le jeu de données $\mathcal{R}$ sera utilisé tout au long de cette section afin d'illustrer les diverses approches présentées pour la prise en compte des éléments manquants.

2.3.1 Normalisation par Projection

Nous décrivons ici un processus utilisé pour normaliser les jeux de données incomplets : la projection. Cette approche est utile dans le cas d'utilisation où l'on cherche à calculer un consensus pour lequel la présence en son sein d'un élément manquant à au moins un des classements d'entrée n'a pas de sens.

Considérons par exemple que l'on interroge plusieurs moteurs de recherche avec la même requête, et que l'on conserve les 1000 premiers résultats. On souhaite ensuite produire un unique classement, un consensus entre ces listes de sites web. Ces listes ne contenant pas nécessairement les mêmes sites web, comment devons-nous considérer un site manquant lorsque l'on va calculer le consensus des classements ? Si on considère qu'un site web non retourné dans les 1000 premiers résultats par un moteur de recherche est nécessairement non pertinent et ne devrait donc pas être présent dans le consensus final, alors la *projection* est l'approche de normalisation à utiliser.

La normalisation d'un jeu de données par projection [BBN13] est une normalisation simple à mettre en œuvre, elle revient à retirer d'un jeu de données tous les éléments absents d'au moins un classement. Si le jeu de données considéré est constitué de permutations (classements sans égalité) alors sa normalisation sera aussi constituée de permutations, et donc utilisable dans le cadre du problème d'agrégation de classements sans égalité.

Exemple 2.13. Soit $\mathcal{R}_P$ le jeu de données produit après projection du jeu de données $\mathcal{R}$ (*cf.* Exemple 2.12) et soit r_P^* le consensus optimal de $\mathcal{R}_P$ au sens du score de Kemeny (*cf.* Définition 1.18 p.21) :

$$\mathcal{R}_P = \begin{pmatrix} [\{A\},\{D\}] \\ [\{A\},\{D\}] \\ [\{D\},\{A\}] \\ [\{A\},\{D\}] \end{pmatrix} \qquad \begin{array}{l} r_P^* = [\{A\},\{D\}] \\ K^{(1)}(r_P^*, \mathcal{R}_P) = 1 \end{array}$$

L'exemple ci-dessus illustre le fort pouvoir filtrant de la projection d'un jeu de données : le consensus obtenu suite à la projection du jeu de données ne contient plus que 40% des éléments initiaux.

2.3.2 Normalisation par Unification

Nous décrivons ici un second processus utilisé pour normaliser les jeux de données incomplets : l'unification. Cette approche est utile dans le cas d'utilisation où l'on cherche à avoir un consensus sur l'ensemble des éléments (par opposition à la

projection), et où un élément absent est considéré comme étant moins pertinent qu'un autre présent au sein d'un classements.

Considérons par exemple que l'on a des moteurs de recherche triant des gènes par ordre de pertinence par rapport à la maladie utilisée pour faire la recherche. On souhaite calculer un consensus entre les différents moteurs de recherche. Si un gène n'est pas retourné par un des moteurs de recherche, comment devons-nous considérer le gène lorsque l'on va calculer le consensus des classements ? Si on considère qu'un gène absent d'un classement signifie qu'il est moins pertinent que ceux retourné, et qu'il doit être présent dans le consensus final, alors l'*unification* est l'approche de normalisation à utiliser.

Le processus d'unification [CBDH11 ; SZ09] produit un jeu de données où pour chaque classement les éléments manquants sont classés après ceux présents dans un groupe *d'unification*. Ce processus produisant un jeu de données complet, il a l'avantage de ne pas avoir besoin de redéfinir le score de Kemeny généralisé pour permettre de produire un consensus, et est donc compatible avec l'ensemble des algorithmes déjà formulés pour le problème de l'agrégation de classements.

Dans [AM12], l'unification est utilisée sur des permutations. Afin de produire des permutations, les auteurs fragmentent arbitrairement les égalités des groupes d'unification en suivant l'ordre alphanumérique entre chacun des éléments.

Exemple 2.14. Soit $\mathcal{R}_U$ le jeu de données produit après unification du jeu de données $\mathcal{R}$ (*cf.* Exemple 2.12), et soit r_U^* le consensus optimal de $\mathcal{R}_U$ au sens du score de Kemeny :

$$\mathcal{R}_U = \begin{pmatrix} [\{A\},\{D\},\{E\},\{B,C\}] \\ [\{A\},\{E\},\{C\},\{D\},\{B\}] \\ [\{B\},\{D\},\{A\},\{C\},\{E\}] \\ [\{A\},\{D\},\{B,C,E\}] \end{pmatrix} \qquad \begin{array}{l} r_U^* = [\{A\},\{D\},\{E\},\{B,C\}] \\ K^{(1)}(r_U^*,\mathcal{R}_U) = 11 \end{array}$$

En comparant le consensus produit ci-dessus par rapport à celui produit suite à la projection du jeu de données (*cf.* Exemple 2.13), on remarque que l'ordre des éléments en commun est respecté, et enrichi des éléments parfois absents grâce à l'interprétation de leur absence. En plaçant ces éléments dans un dernier groupe, ils deviennent explicitement moins pertinents que l'ensemble des autres éléments du classement.

2.3.3 Définition de la mesure de Kendall-τ induite et du score de Kemeny induit

Nous décrivons ici l'adaptation d'une distance afin de travailler avec des jeux de données incomplets : la mesure de Kendall-τ induite [Dwo+01]. Cette adaptation permet de ne rien supposer de l'absence d'un élément, et de calculer un consensus sur l'ensemble des éléments apparaissant dans le jeu de données. Nous avons généralisé [2] cette mesure dans [Sta+14], afin de mettre en œuvre un droit de réserve.

Considérons que des experts d'un domaine classent des éléments par ordre de similarité par rapport à un élément de référence. Ces experts n'ayant pas l'occasion de confronter leurs avis afin de produire un consensus, ce dernier doit être calculé. Si un expert s'est abstenu de classer un des éléments candidat par rapport à l'élément de référence, comment devons-nous le considérer lorsque l'on va calculer le consensus des avis d'experts ? Considérer que cet élément ne doit pas être dans le consensus (projection) est incorrect. Il est aussi incorrect de considérer que l'expert perçoit l'élément manquant comme étant moins intéressant que les autres, car l'expert ne s'est justement pas exprimé sur cet élément. Si on veut considérer uniquement l'ordre entre chaque paire d'éléments tels qu'exprimé dans le jeu de données et ne rien supposer de l'absence d'éléments dans un classement, alors il faut utiliser la mesure Kendall-τ induite.

2.3.3.1 Mesure de Kendall-τ induite

La mesure de Kendall-τ induite adapte la distance de Kendall-τ afin d'accepter en entrée des classements incomplets. Pour ce faire, la mesure ne considère entre deux classements que les paires d'éléments communs au deux classements.

Définition 2.1. La mesure de Kendall-τ induite, notée ici J, est définie entre deux permutations $\pi \in \mathbb{S}_n^I$ et $\sigma \in \mathbb{S}_n^I$ de la façon suivante :

$$J(\pi,\sigma) = |\{(i,j) \in elem(r) \cap elem(s) : i < j \wedge \\ (\pi[i] < \pi[j] \wedge \sigma[i] > \sigma[j] \vee \pi[i] > \pi[j] \wedge \sigma[i] < \sigma[j])\}|$$

où la fonction $elem(\pi)$ retournant l'ensemble des éléments classés dans π tel que défini en Section 1.5.1 (p. 23).

2. La généralisation de la méthode est introduite au Chapitre 3, son contexte d'application qui est le consensus d'avis d'expert en considérant les égalités est lui détaillés au Chapitre 5.

Exemple 2.15. Soit $\pi = [A, B, D, E]$ et $\sigma = [A, B, F, E, D]$, $J(\pi, \sigma) = 1$. En effet $elem(\pi) \cap elem(\sigma) = \{D, E\}$, l'unique paire d'éléments sur laquelle on peut constater un désaccord :$J(\pi, \sigma) = 1$.

L'adaptation de la distance est une mesure. La mesure de Kendall-τ induite est nommée "*distance de Kendall-τ induite*" dans [Dwo+01], cependant ce n'est pas une distance (*cf.* Définition 1.2 p.13) car elle ne respecte ni l'inégalité triangulaire (*cf.* Équation 2.3), ni l'identité (*cf.* Équation 2.4).

$$\text{Soit } \pi_1 = [B, A], \pi_2 = [A, B], \pi_3 = [A, C]$$
$$J(\pi_1, \pi_2) = 1 \not\leq J(\pi_1, \pi_3) + J(\pi_3, \pi_2) = 0 \tag{2.3}$$
$$J(\pi_1, \pi_3) = 0 \text{ alors } \pi_3 \text{ est différent de } \equiv \pi_1 \tag{2.4}$$

Sur un plan pratique le fait que ce ne soit pas une distance permet d'exprimer ce que nous avons qualifié de "droit de réserve". Dans l'exemple 2.16, les permutations de $\mathcal{B}$ sont différentes les uns des autres sans pour autant avoir de désaccords entre elles (ni directement ni par transitivité), et l'utilisation de la mesure de Kendall-τ induite permet de (re-)construire un consensus à partir des informations parcellaires que donnent les permutations.

Exemple 2.16. Soit $\mathcal{B} \subset \mathbb{S}_6$, et soit π^* le consensus optimal de $\mathcal{A}$ au sens du score de Kemeny induit (défini ci après comme la somme des distances de Kendall-τ induite) :

$$\mathcal{Z} = \begin{pmatrix} [A, B, D, E, F] \\ [B, C, F] \\ [C, D, F] \end{pmatrix} \qquad \pi^* = [A, B, C, D, E, F]$$

2.3.3.2 Score de Kemeny induit

Le score de Kemeny induit d'une permutation π vis-à-vis d'un ensemble de permutations en entrée $\mathcal{P}$ est construit comme étant la somme des distances de Kendall-τ induites entre π et les permutations de $\mathcal{P}$.

Définition 2.2. Pour tout ensemble de permutations $\mathcal{P} \subseteq \mathbb{S}_n^I$ et une permutation $\pi \in \mathbb{S}_n^I$, le score de Kemeny, noté ici S^I, est :

$$S^I(\pi, \mathcal{P}) = \sum_{\sigma \in \mathcal{P}} J(\pi, \sigma)$$

Exemple 2.17. Le jeu de données $\mathcal{R}$ introduit en Exemple 2.12 n'est pas altéré en utilisant le mesure de Kendall-τ induite, un des consensus au sens du score de Kemeny induit est r_I^* :

$$\mathcal{R} = \begin{pmatrix} r_1 = [\{A\},\{D\},\{E\}] \\ r_2 = [\{A\},\{E\},\{C\},\{D\}] \\ r_3 = [\{B\},\{D\},\{A\},\{C\}] \\ r_4 = [\{A\},\{D\}] \end{pmatrix} \qquad \begin{array}{l} r_I^* = [\{B\},\{A\},\{E\},\{C\},\{D\}] \\ S^I(r_I^*,\mathcal{R}) = 3 \end{array}$$

En comparant le consensus produit suite à l'unification du jeu de données, celui produit suite à sa projection, ou encore celui produit en utilisant la mesure de Kendall-τ induite, on remarque que l'interprétation différente de l'absence des éléments produit des consensus radicalement différents. L'élément B est présent dans un seul classement du jeu de données original en première position, il en résulte qu'il est (i) premier en utilisant la mesure de Kendall-τ induite car les autres classements "s'abstiennent" alors qu'il est (ii) dernier en utilisant l'unification car son absence des autres classements fait qu'il est considéré comme non-pertinent, et finalement (iii) absent en utilisant la projection du fait qu'il est considéré comme non-pertinent.

2.3.4 Discussion

Le problème de l'agrégation de classements avec égalités, lorsque l'on le formalise avec le score de Kemeny, ne permet pas de prendre en entrée des classements incomplets. Nous avons présenté dans cette section trois approches pour prendre en compte des jeu de données incomplets en interprétant différemment l'absence d'un élément. Leurs interprétations radicalement différentes des données, comme l'ont montré les différents exemples sur un même jeu de données, amènent à conclure que des résultats obtenus sur les mêmes données mais avec des normalisations différentes seront au mieux difficilement comparables, et non-nécessairement transposables d'une normalisation à l'autre.

2.4 Précédents résultats

Dans cette section nous synthétisons les résultats obtenus par les précédentes études comparatives rencontrées à ce jour dans la littérature. Après avoir présenté une mesure pour évaluer la qualité d'un résultat, nous présentons (2.4.2) les conclusions de la seule étude produisant des classements avec égalités, nous présentons ensuite (2.4.3) les conclusions formulées sur des données réelles, puis (2.4.4) sur

des données synthétiques, dans un troisième temps (2.4.5) nous nous concentrons sur les différentes approches de normalisation utilisées pour finalement conclure.

2.4.1 Une mesure pour évaluer la qualité des résultats

Pour comparer la qualité de deux consensus, deux approches peuvent être suivies. La première est d'utiliser la distance réelle de chaque consensus par rapport aux classements en entrée [CW09]. La seconde approche nommée *gap* [AM12 ; SZ09] consiste à normaliser la distance de chaque consensus par la distance du consensus optimal. Cette normalisation permet de montrer les désaccords supplémentaires qu'une solution a par rapport à une solution optimale.

Définition 2.3. Soit c^* un consensus optimal, soit c un quelconque consensus retourné par un algorithme donné pour un jeu de données $\mathcal{R}$. Le *gap* est défini comme suit :

$$gap = \frac{K(c\ ,\mathcal{R})}{K(c^*,\mathcal{R})} - 1$$

On note qu'un consensus optimal a un *gap* de 0.

2.4.2 Résultats avec des égalités en entrée et en sortie

La seule étude expérimentale ayant évalué les approches en considérant à la fois les égalités en entrée et la production de classements avec des égalités est [CBDH11]. Les auteurs ont évalué quatre approches (Ailon$\frac{3}{2}$, BioConsert, FaginDyn et RepeatChoice) et concluent de leurs expérimentations que BioConsert est le plus performant en qualité. Cependant aucune indication quant au temps de calcul n'est fournie, et les résultats ont été obtenus sur un nombre réduit de jeu de données réels (11 jeux de données biomédicaux contenant entre 15 et 402 éléments) et sur de très petits jeux de données générés uniformément (des jeux de données contenant que de 4 à 8 éléments).

Les algorithmes mais aussi les jeux de données considérés dans [CBDH11] diffèrent totalement de ceux utilisés dans d'autres études [AM12 ; BBN13 ; CW09 ; SZ09] qui sont décrites ci-après.

2.4.3 Résultats sur des données réelles

Les auteurs de [AM12] et [SZ09] ont comparé des approches prenant en entrée des classements avec égalités, mais étant basées sur la distance de Kendall-τ

Nom et publication		Sur les même éléments	n	m	#
EachMovie	[CW09]	non	100		146
F1	**[BBN13]**	projeté	$[9; 28]$	$[11; 19]$	39
BioMedical	**[CBDH11]**	unifié	$[15; 402]$	4	11
GiantSlalom	[AM12]	unifié fragmenté	59	16	1
SkiCross/Jumping	**[BBN13]**	projeté	69	4	1
WebCommunities	[CW09 ; SZ09]	oui	100	9	50
WebSearch	[Dwo+01]	–		7	37
	[SZ09]	unifié	$\overline{n} = 283$	4	37
	[AM12]	unifié	$[275; 348]$	4	37
	[BBN13]	projeté	$[18; 163]$	4	37
Mallows model	[AM12]	oui	$\{10, 50\}$	$\{100, 5000\}$	8
Mallows model	[BBN13]	oui	$[10; 200]$	$[4; 20]$	8400
Placket-Luce model	[AM12]	oui	$\{10, 50\}$	100	2
	[BBN13]		$[10; 200]$	$[4; 20]$	2240
RandomGraph	[DK04]	oui	$[15 - 50]$	$[5; 35]$	
RandomGraph	[CW09]	oui			1500
Random	[AM12]	oui	100	100	1
Random	[CBDH11]	oui	$[4; 8]$	4	2500

TABLE 2.3 – Jeux de données utilisés dans la littérature et approches de normalisation suivi. Pour chaque jeux de données on fournit la taille des jeux de données en terme d'éléments (n), de classements (m) et de nombre de jeux de données utilisés (#). Les auteurs ayant rendu disponible leur jeux de données sont indiqués via une référence bibliographique en gras.

classique. Ces approches produisent nécessairement des permutations et sont incapables de prendre en compte le coût de créer/rompre des égalités. Dans [AM12], les auteurs ont utilisé les jeux de données WebSearch et GiantSlalom alors que dans [SZ09], les auteurs ont travaillé sur le jeux de données WebSearch et WebCommunities. De leurs expérimentations respectives ils concluent que :

1. Chanas produit des résultats de haute qualité (les résultats ont un *gap* de $0,05\%$, il y a donc moins de 0.05% de désaccord supplémentaires par rapport à l'optimum [AM12]).
2. Les approches positionnelles (comme la méthode de Borda et celle de Copeland) sont des approches capables de fournir très rapidement des résultats

de qualité correcte (à 2.5% de l'optimum [AM12 ; SZ09]).

3. KwikSort offre un bon compromis entre les recommandations précédentes.
4. Une approche intermédiaire entre KwikSort et ChanasBoth est possible pour [AM12] en utilisant leur approche en *branch-and-bound* (*cf.* Exemple 2.2.2.2) avec les techniques de recherche par faisceau. Cette technique permet lors de l'exploration de l'arbre de ne pas explorer l'ensemble des branches, mais uniquement les plus prometteuses.

Dans [AM12], CopelandMethod renvoie de meilleurs résultats que BordaCount, ceci est uniquement dû à la présence d'égalités dans les classements. En effet, dans le cas de permutations les résultats sont nécessairement équivalents [SZ09]. Quant à CopelandMethod et MC4, dans [SZ09] ils présentent des résultats comparables en termes de qualité, MC4 étant beaucoup plus coûteux en temps.

Dans [CW09], les auteurs se concentrent seulement sur les permutations en utilisant les jeux de données WebCommunities et EachMovie. Par rapport à [AM12 ; SZ09], ils confirment que Chanas et ChanasBoth produisent des résultats de qualité mais ils sont en désaccord complet sur l'utilisation de KwikSort (qui obtient avec [CW09] une mauvaise performance). Les algorithmes positionnels ne sont pas considérés dans cette étude.

Le calcul du consensus optimal est étudié par [BBN13] en utilisant la Programmation en Nombres Entiers (PNE). Après avoir identifié une propriété théorique centrée sur les éléments permettant de découper le problème en deux sous-problèmes indépendants (*cf.* Section 2.2.2.3), les auteurs évaluent leur pré-traitement sur des données réelles et synthétiques afin d'évaluer le découpage effectif des jeux de données et le gain en temps de calcul. Dans leur étude, ils comparent trois logiciels utilisés (GLPK, CPLEX et Gurobi) pour résoudre des PNE, et conclut que GLPK est toujours le plus lent des trois. L'ensemble des données présentes dans l'article indique de plus que CPLEX est toujours plus rapide que Gurobi.

2.4.4 Résultats sur des données synthétiques

En plus des données réelles, quelques jeux de données générés ont été considérés. Une approche pertinente dans [AM12] a été de générer des jeux de données avec différents niveaux de similarité. On qualifie cette similarité de *a priori* car elle est mesurable uniquement via les paramètres de génération,

BordaCount semble être le meilleur choix lorsque la similarité *a priori* est forte tandis que l'approche en *branch-and-bound* devrait être préférée lorsque la similarité *a priori* est faible. De plus, les auteurs de [AM12] confirment que ChanasBoth propose des solutions de grande qualité. Les auteurs de [DK04], [BBN13]

et [AM12] observent que le temps requis pour calculer un consensus optimal décroît lorsque la similarité entre les permutations augmente.Aucune similarité n'est mesurée *a posteriori* sur les données après génération.

Les auteurs de [AM12] distinguent ensuite des concepts de "fort consensus" et "faible consensus" en fonction des paramètres utilisés dans la génération des données synthétiques avec similarité, ils ne relient pas la notion de similarité à une similarité mesurable. On rappel que les auteurs de [AM12 ; BBN13 ; DK04] ne fournissent en effet qu'une similarité *a priori*. L'absence de mesure pour évaluer numériquement la similarité d'un jeu de données quelconque rend alors l'application des différentes recommandations des auteurs difficiles pour des cas concrets.

Des jeux de données uniformément générés sont utilisés par [AM12 ; CSS99], cependant aucune conclusion nouvelle n'est obtenu de ces études qui n'aient déjà été mises en évidence sur les jeux de données réels.

2.4.5 Normalisation des jeux de données

Différentes normalisations sont utilisées dans la littérature. [BBN13] normalisent les jeux de données en utilisant la projection (*cf.* Section 2.3.1 p.38) tandis que [SZ09 ; CBDH11] utilisent l'unification (*cf.* Section 2.3.2 p.38). [AM12] utilise aussi l'unification, mais pour les jeux de données GiantSlalom, ils fragmentent les égalités en départageant les compétiteurs à égalité en fonction de leur nom de famille. Savoir si le choix de normalisation à un impact sur les résultats produits est une question qui reste ouverte.

2.4.6 Bilan

Pour résumer, chaque étude a considéré un ensemble (restreint) d'algorithmes donnés et a effectué des expérimentations sur des jeux de données variés, créés avec diverses méthodes, et produisant pour la plupart des permutations en sortie. Compte tenu des résultats actuels fragmentaires, voir contradictoires, pouvant être extraits de la littérature, il est très difficile de déterminer dans quel contexte un algorithme, ou une approche de normalisation, devrait être préféré(e) à un(e) autre.

2.5 Conclusion

Dans ce chapitre nous avons (1) présenté plusieurs algorithmes en distinguant s'ils considéraient ou non les égalités dans les classements et le coût de créer/-rompre ces égalités, mais aussi s'ils se concentraient sur les désaccords entre paires d'éléments ou sur la position des éléments dans les classements. Nous avons ensuite (2) présenté trois approches disponibles dans la littérature (deux normalisations

et une adaptation d'une distance) pour faire le lien entre des jeux de données *incomplets* (dans lesquels certains éléments ne sont pas présents dans tous les classements) et le cadre de l'agrégation de classements qui est défini pour des classements *complets*. Nous avons listé quelles approches avait été appliquées sur des jeux de données disponibles dans la littérature. Nous avons finalement (3) présenté une vue globale des résultats obtenus sur des données synthétiques et réelles en indiquant les tailles et volumes de données pris en compte.

Nous avons vu que seuls deux algorithmes considèrent pleinement les égalités et les désaccords liés à la présence d'égalité alors qu'une large majorité des algorithmes sont appliqués sur des données avec des égalités, négligeant ainsi les désaccords liés aux égalités. Nous verrons dans le chapitre suivant comment évaluer les algorithmes afin de pleinement prendre en compte les égalités, et nous aborderons les questions quant à la complexité que soulèvent les égalités

Nous avons vu que les approches disponibles pour travailler avec des jeux de données incomplets produisent des consensus très différents. Le contexte dans lequel est appliquée l'agrégation de classements et les besoins réels liés à son application influent sur le choix parmi les différentes approches de normalisations. Nous verrons dans le chapitre suivant de nouvelles approches pour travailler avec des jeux de données incomplets découlant de différents besoins réels et présenterons une méthode pour guider l'utilisateur dans son choix entre les approches en fonction de son besoin.

Nous avons vu que les résultats présents dans la littérature sont issus d'algorithmes qui ne gèrent pas les égalités sur des jeux de données avec égalités, qu'ils contiennent des contradictions, et que les tailles et volumes de données considérées peuvent être insuffisants pour dresser des conclusions. Nous avons aussi vu que les recommandations quant à la similarité ne sont pas applicables dans des cas réels. L'ensemble de ces problèmes seront adressés dans le Chapitre 4 afin de fournir une vue claire des performances des algorithmes en qualité et en temps en fonction de différentes caractéristiques des jeux de données. Ces évaluations permettront finalement de répondre au besoin de recommandations réutilisables.

Résumé du chapitre 2

Dans ce chapitre nous avons présenté différents algorithmes pour calculer des consensus au sein du problème de l'agrégation de classements. Pour ces heuristiques et algorithmes d'approximation, nous avons proposé une classification en fonction de l'information de base exploitée pour calculer un consensus, à savoir les désaccords entre paires d'éléments (avec et sans prise en compte des égalités) ou encore la position des éléments. Nous avons de plus indiqué à quelle classe d'algorithme ils appartiennent, et avons illustré leur fonctionnement par un exemple sur un jeu de données commun. Nous avons finalement noté que seul deux algorithmes considèrent pleinement les égalités et les désaccords liés à leur présence.

Nous avons ensuite présenté plusieurs stratégies permettant de répondre à une problématique présente avec les données réelles : le fait que les jeux de données réels ne contiennent pas nécessairement les mêmes éléments (ils sont *incomplets*) alors que le problème de l'agrégation de classements impose d'avoir des jeux de données où les classements ont les mêmes éléments. Nous avons présenté deux processus (unification et projection) et une distance permettant de faire le lien entre les données réelles et le formalisme du problème (mesure de Kendall-τ induite).

Dans un troisième temps, nous avons synthétisé les résultats obtenus par les précédentes études rencontrées à ce jour dans la littérature. Nous avons décomposé les résultats en trois axes selon que les résultats a été fait en prenant en compte les égalités, en se basant sur des données réelles, ou synthétiques. Nous avons observé que les résultats était difficilement comparables entre les études du fait que les processus de normalisation ne sont pas nécessairement les mêmes, que les jeux de données sur lesquels les résultats ont été produits ne se recoupent pas nécessairement. Nous avons noté que les jeux de données sont rarement rendus disponibles. Nous avons finalement conclu que les résultats sont trop fragmentaires, voire contradictoires, pour déterminer quel algorithme et quelle approche de normalisation devraient être préférés dans une problématique réelle.

Chapitre 3

Besoins rééls, évolution des algorithmes et complexité

Sommaire

3.1 Introduction

Le chapitre précédent a fait état d'un grand nombre d'algorithmes existants pour calculer un consensus de classements. Le choix entre ces différents algorithmes dépend des cas d'utilisations et besoins auxquels il faut répondre. En particulier, la distance ou la mesure qui doit être utilisée pour comparer les classements doit être choisie avec soin. Le choix du processus de normalisation des données est un second point particulièrement important. Un troisième point important est d'apporter des réponses quant à la complexité du problème de l'agrégation de classements, et en particulier lorsque l'on considère les égalités entre éléments. L'objectif de ce chapitre est alors triple.

Premièrement, nous souhaitons identifier clairement des caractéristiques clés des jeux de données ayant un impact sur la façon de considérer ces données. Par exemple, *"Les classements sont-ils ou non sur les mêmes données ?"*, ou encore, *"En cas d'élément manquant dans l'un des classements, préfère-t-on ne plus considérer cet élément dans l'ensemble des classements ou au contraire compléter les classements pour qu'ils considèrent tous les mêmes éléments ?"*. Nous proposons alors un *guide à l'utilisateur* dans lequel, selon ces critères, nous conseillons l'utilisateur dans le choix du processus de normalisation à utiliser et de la distance (ou mesure) à privilégier. Ce guide fera apparaître deux mesures qui n'ont pas encore été introduites et qui le seront donc dans ce chapitre.

Deuxièmement, l'identification des mesures, distances et processus de normalisation à utiliser mène ensuite au choix d'un algorithme utilisant ces outils. Une particularité des données qui apparaît souvent (naturellement présente dans les données ou suite à une normalisation) est la présence d'égalités entre éléments, c'est-à-dire, le fait que des éléments soit classés *ex-æquo*. Nous avons vu au chapitre précédent que peu d'algorithmes permettent de calculer des consensus avec des égalités. Le second objectif de ce chapitre est alors d'étudier de façon approfondie le problème du consensus de classements dans le cas de classements avec égalités. En particulier, nous adapterons plusieurs algorithmes afin de prendre en compte les égalités et, quand c'est possible, les désaccords liés au fait de créer/-rompre des égalités entre éléments. Nous proposerons aussi un nouvel algorithme permettant de calculer la(les) solution(s) exacte(s) d'une instance quelconque du problème de l'agrégation de classements avec égalités. Finalement, nous proposerons des améliorations réduisant la complexité de BioConsert, un algorithme prenant nativement en compte les égalités entre éléments.

Troisièmement, le problème de l'agrégation de classements lorsque l'on ne considère pas les égalités est connu pour être NP-difficile [Dwo+01 ; BBD09] pour un nombre pair de permutations plus grand que trois. Ce résultat de complexité ne

couvre ni le cas où les classements possèdent des égalités entre éléments, ni celui où les classements sont incomplets. Nous étudierons donc la complexité de ces problèmes nouvellement formalisés.

Ce chapitre s'organise de la façon suivante. Premièrement, nous présentons un guide pour l'utilisateur permettant d'identifier clairement les particularités des données et de choisir les mesures, distances et normalisations adéquates (*cf.* Section 3.2). Dans un second temps, nous adaptons les algorithmes introduits au chapitre précédent pour qu'ils puissent tenir compte d'une particularité très présente dans les jeux de données réels : l'égalité entre éléments dans des classements (*cf.* Section 3.3). Dans un troisième temps, nous proposons un nouvel algorithme exact pour calculer des consensus optimaux en présence d'égalité entre les éléments (*cf.* Section 3.4). Dans un quatrième temps nous présentons une amélioration permettant de réduire la complexité de l'algorithme BioConsert (*cf.* Section 3.5). Finalement, nous étudions la complexité du problème de l'agrégation de classements dans le cas des classements avec égalités et des classements incomplets (*cf.* Section 3.6).

3.2 Choix des distances et des méthodes de normalisation en fonction de problématique réelles

Nous cherchons dans cette section à faire le lien entre des problématiques réelles et un cadre formel pour lequel il existe des algorithmes pour calculer des consensus. Considérons une recherche de gènes "pertinents" pour une maladie donnée. Faire cette recherche sur deux moteurs de recherche biomédicaux différents ne va pas nécessairement renvoyer des listes contenant les mêmes éléments, il est de plus possible que plusieurs gènes soient considérés comme ayant le même niveau de pertinence et donc soient à égalité (si par exemple ils appartiennent à la même voie métabolique). Face à cette problématique réelle, le cadre formel de l'agrégation de classements tel que défini dans [Dwo+01] considère que les classements ne contiennent pas d'égalité, et qu'ils sont complets (dans un jeu de données chaque classement contient les mêmes éléments).

Nous avons présenté en Section 2.3 plusieurs approches pour travailler avec des jeux de données incomplets, à savoir deux normalisations du jeu de données et une adaptation de la distance de Kendall-τ. Afin de placer l'ensemble de ces approches dans un cadre global se concentrant sur les besoins qu'ont les utilisateurs voulant obtenir un consensus entre plusieurs classements, nous caractérisons le besoin dans un *guide pour l'utilisateur* qui, via une succession de choix, débouche sur l'utili-

sation de variantes de la distance de Kendall-τ et des processus de normalisation. Ce *guide pour l'utilisateur* permet aussi de mettre en lumière qu'il y a d'autres besoins auxquels l'état de l'art en matière de distance et de normalisation n'a pas encore répondu.

Dans cette section, nous présentons le guide et illustrons son utilisation, puis nous introduisons la distance de Kendall-τ généralisée induite, une variante qui permet de comparer deux classements même s'ils ne contiennent pas les mêmes éléments. Dans un troisième temps, nous introduisons une nouvelle pseudo-distance qui, comme il le sera montré, permet une interprétation plus fine, que l'état de l'art ne le permet actuellement, de l'absence d'éléments dans certains classements, et présent dans d'autres.

3.2.1 Caractérisation des besoins des utilisateurs

Afin d'identifier clairement quel processus doit être appliqué et quelle distance ou mesure doit être utilisée lorsque les classements ne contiennent pas exactement les mêmes éléments et/ou contiennent des éléments à égalité, nous présentons en Figure 3.1 un guide pour aider au choix d'une normalisation et d'une distance.

Nous expliquons dans un premier temps les choix et questions que ce *guide à l'utilisateur* contient. Dans un second temps nous illustrons l'intérêt d'un tel guide au travers de quatre exemples basés sur l'utilisation de moteurs de recherche.

3.2.1.1 Un guide à l'utilisateur pour caractérisé ses besoins

Le guide présenté en Figure 3.1 part d'un jeu de données puis, via une succession de questions, établit les besoins en mesure, distance et processus de normalisation. Nous détaillons ci-après les questions

c0 : Ce choix interroge sur la complétude d'un jeu de données, c'est-à-dire si chaque classement du jeu de données contient les mêmes éléments.

c1 : Cette question est posée lorsqu'un jeu de données est incomplet, et permet de caractériser le jeu de données. On cherche à savoir si, au sein d'un classement, les éléments qui sont absents de ce dernier mais présents dans d'autres ont un intérêt. S'ils ont un intérêt, même minime on accède au choix c2, sinon le guide nous informe que le processus de projection doit être utilisé car il va retirer tous les éléments qui manquent à au moins un classement.

c2 : La jeu de données considéré dans cette question est incomplet, et les éléments absents d'un classement ont une importance. On s'interroge ici sur la pertinence d'un élément absent d'un classement par rapport au éléments présents. Si aucune supposition quant à la pertinence peut être faite, alors on accède au choix c4. Dans le cas contraire le processus d'unification est recommandé car il va

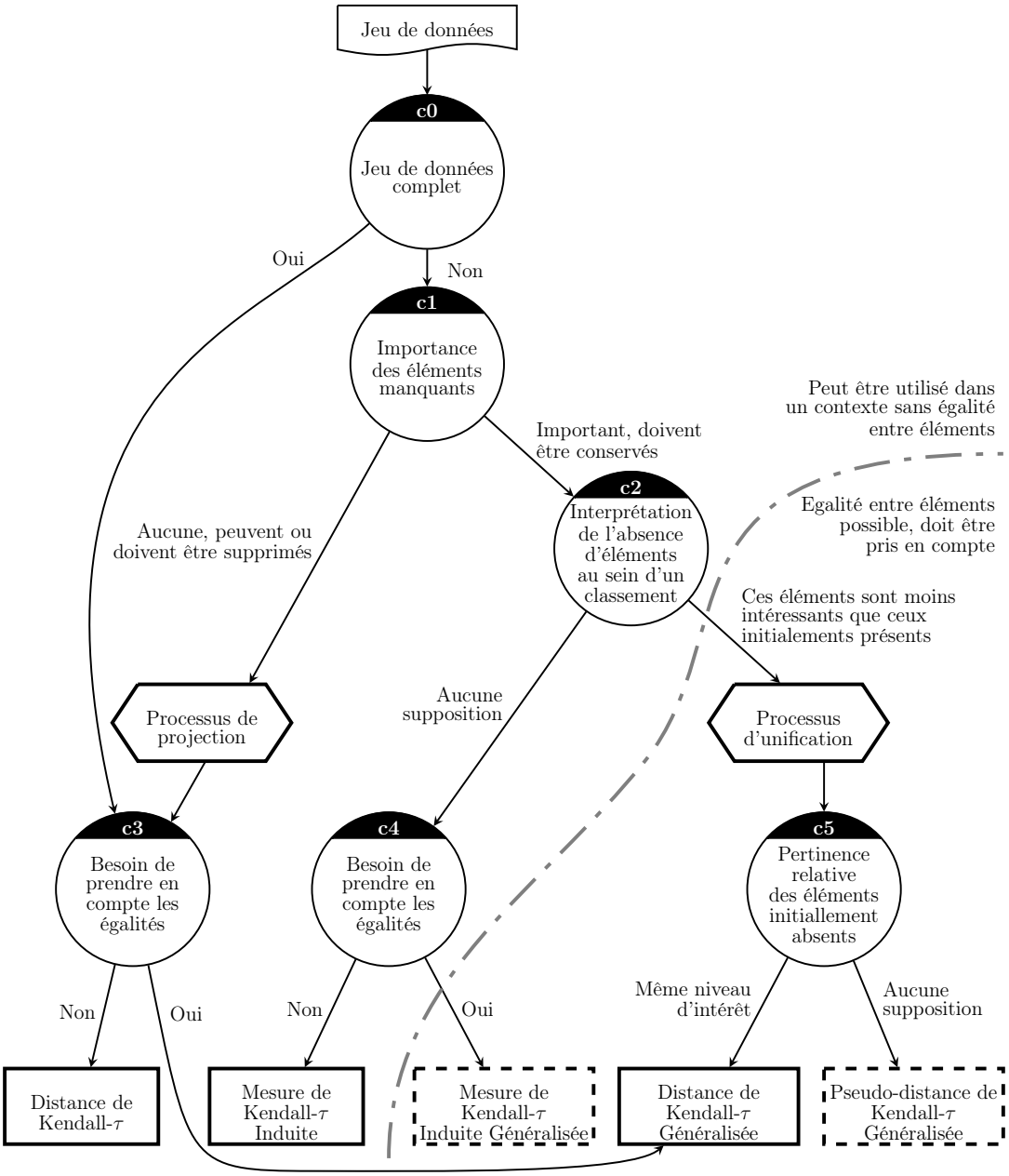

FIGURE 3.1 – Guide pour aider à choisir le processus (ou l'approche) de normalisation des données et la distance (ou mesure) en fonction des caractéristiques du jeu de données et des besoins. Les processus sont dans des hexagones, les distances et mesures dans des rectangles, et les choix dans des cercles. Les deux distances dans des rectangles en pointillé sont absentes de l'état de l'art et introduites dans cette section.

ajouter à la fin de chaque classement les éléments manquants, et ainsi interprété qu'ils sont moins pertinents.

c3/c4 : Ces deux choix nous interrogent sur la présence d'égalités au sein du jeu de données, et donc sur le besoin de les prendre en compte, pour aboutir ensuite à différentes distances et mesures.

c5 : Ce dernier choix permet de préciser l'interprétation de l'absence d'élément d'un classement dans le jeu de données original. On sait à ce stade que ces éléments (initialement) absents sont moins pertinent que les éléments (initialement) présents, on cherche ici à préciser si deux éléments absents d'un même classement ont une pertinence équivalente. En considérant par exemple un moteur de recherche, on cherche ici à savoir si deux éléments non retournés par ce dernier sont strictement non pertinents (réponse "Même niveau d'intérêt"), ou s'il est possible que l'un soit *peu pertinent* alors que l'autre est *strictement non pertinent* (réponse "Aucune supposition").

3.2.1.2 Consensus entre les k premiers résultats de moteurs de recherche

Ce scénario basique reprend celui utilisé pour la génération des jeux de données WebSearch [Dwo+01 ; AM12 ; BBN13]. Dans ce scénario, la même requête est exécutée, par exemple "amusement parks", dans plusieurs moteurs de recherche (Google, Excite, Lycos, ...) pour ensuite calculer un consensus entre les listes de résultats. Le but décrit dans [Dwo+01] est de combattre le spam" c'est-à-dire mettre en avant les sites web retournés par beaucoup de moteurs au détriment de ceux retourné par peu de moteurs de recherche et considéré comme spam. Les listes de résultats ne contiennent pas l'ensemble des sites web retournés par les moteurs de recherche, mais uniquement les 10 premiers. Mettons alors en œuvre notre guide à l'utilisateur afin de savoir quelles mesure, distance et processus de normalisation utiliser.

Le choix c0 de notre guide nous permet d'indiquer que le jeu de données est incomplet, en effet chaque liste contient les 10 premiers sites de chaque moteurs, mais ne sont pas nécessairement les mêmes. Le choix c1 permet d'indiquer que chaque résultat doit être conservé, et en effet un site absent d'un classement peut tout à fait être premier dans les autres classements, et donc être un site web pertinent. Ensuite avec c2 nous précisons tout de même qu'un élément absent d'un classement est moins intéressant que ceux initialement présents dans ce même classement. Cette décision nous amène alors à utiliser le processus d'unification (*cf.* Section 2.3.2) qui ajoute à la fin de chaque classement un groupe dit *d'unification* où sont à égalité les éléments absents du classement mais présent dans d'autres dans le jeu de données. Le choix c5 nous questionne finalement sur la pertinence relative de deux éléments initialement absents d'un même classement. L'état de

l'art actuel nous amène à utiliser la distance de Kendall-τ généralisée, et donc à considérer qu'ils sont à égalité, or cette égalité est artificielle car absente des données originales. Afin de palier nous présentons en Section 3.2.3 une pseudo-distance permettant d'interpréter plus finement les égalités, et de différencier ces égalités artificielles des réelles. Dans notre scénario nous ignorons si deux éléments absents ont la même (non-)pertinence.

On conclut alors que notre guide à l'utilisateur nous a permis de déterminer qu'il fallait appliquer le processus d'unification sur les données avant de calculer le consensus en se servant de la pseudo-distance de Kendall-τ généralisée. En effet le processus d'unification crée des égalités en éléments qui doivent être interprétées finement.

3.2.1.3 Consensus entre de nombreux résultats de moteurs de recherche

Reprenons le scénario utilisé précédemment en changeant un point : on conserve ici pour chaque moteur de recherche les 100 premiers résultats (au lieu de 10). Vu que l'on conserve les 100 premiers résultats de chaque moteur de recherche, et que généralement un utilisateur ne regarde que les 10 ou 20 premiers résultats, on considère ici qu'un site web non retourné dans les 100 premiers résultats par un moteur de recherche est nécessairement non pertinent et ne devrait donc pas être présent dans le consensus final. Suivons donc de nouveau notre guide à l'utilisateur pour déterminer les distance et processus à utiliser.

Le choix c0 permet d'indiquer ici encore qu'un jeu de données est incomplet. Lors du choix c1, on indique ici qu'un résultat absent d'un classement n'est pas pertinent (au mieux 101^e) et doit être retiré. Le guide nous indique alors que l'on doit utiliser le processus de projection (*cf.* Section 2.3.1), un processus qui va retirer d'un jeu de données les éléments absents d'au moins un des classements. Viens enfin le choix c3, les moteurs de recherche généralistes ne permettant pas de savoir si deux éléments sont à égalité, notre guide nous indique donc qu'il faut utiliser la distance de Kendall-τ dans sa formulation classique.

3.2.1.4 Consensus entre les résultats de moteurs de recherche biomédicaux

Nous considérons dans ce scénario encore des moteurs de recherche, mais dédiés à l'exploration des données contenues dans des sources biologiques et biomédicales. Certains moteurs comme BioZone [BY06 ; SIY06] peuvent présenter les résultats comme le ferait un moteur de recherche généraliste (à noter que BioZone est justement basé sur PagraRank[Pag+99], l'algorithme de Google). D'autres moteurs tel que BioGuideSRS [CB+07] exploitent les liens typés présents entre les données

biologiques et présentent les résultats en fonction des chemins d'exploration qui ont mené à chaque résultat, ainsi que la fiabilité et complétude de chaque source intermédiaire.

Considérons que nous utilisons BioGuideSRS et avons plusieurs algorithmes pour classer les résultats obtenu via l'exploration des sources par les *chemins* de BioGuideSRS.

Le choix c0 de notre guide nous permet d'indiquer que le jeu de données est complet, en effet chaque algorithme de classement considérera les mêmes chemins, et donc les mêmes données biologiques. Le choix c3 nous interroge ici sur la possible présence d'égalités entre éléments. Il est en effet possible que plusieurs gènes impliqué dans la même voie métabolique soient pointé par le même ensemble de chemins, dans ce cas les algorithmes pourrait ne pas pouvoir les discriminer, amenant à la présence d'égalité qui ont un sens biologique. Le choix c3 nous indique donc qu'il faut utiliser la distance de Kendall-τ généralisée. On remarque qu'aucun processus de normalisation n'a été recommandé dans ce scénario.

3.2.1.5 Consensus entre experts

Nous quittons les scénarios des moteurs de recherche pour reprendre l'exemple d'utilisation présenté en Section 2.3.3. Dans cet exemple des experts classent par rapport à un élément de référence plusieurs éléments en fonction de la similarité qu'ils perçoivent vis-à-vis de l'élément de référence. On rappelle que, dans cet exemple, un expert peut s'abstenir de classer un élément s'il n'est pas sûr de son opinion. Nous enrichissons cet exemple en indiquant qu'un expert peut estimer que deux éléments ont le même niveau de similaires vis-à-vis de l'élément de référence, et qu'ils sont dans ce cas à égalité dans le classement.

Utilisons notre guide pour identifier quelles mesures utiliser. Par le choix c0, nous indiquons que le jeu de données est incomplet. Le choix c1 permet d'indiquer que tous les éléments sont importants, en effet chaque élément représente l'opinion d'une personne considérée comme experte du domaine étudié et mérite donc une attention. Le choix c2 nous permet d'indiquer une particularité de notre scénario : lorsqu'un élément est manquant d'un classement c'est une abstention d'un expert, et on ne doit rien supposer quant à la pertinence de l'élément. Le guide à l'utilisateur nous oriente alors vers le choix c4. Comme précisé précédemment, il y a des égalités entre éléments, le guide nous amène donc à utiliser la généralisation de la mesure de Kendall-τ induite. Cette mesure n'est pas présente dans la littérature et nous l'introduisons en Section 3.2.2.

3.2.1.6 Discussion autour des cas d'utilisation

Nous avons présenté dans cette sous-section quatre cas d'utilisation débouchant sur des normalisations et distances/mesures différentes. Certaines, déjà présentes dans la littérature, ont été introduites au Chapitre 2. Nous présentons maintenant deux nouvelles mesures répondant à de nouveaux besoins identifiés grâce à notre *guide à l'utilisateur*.

3.2.2 Une mesure pour un droit de réserve et des égalités entre éléments

La mesure de **Kendall-τ généralisée induite** combine des propriétés de la distance de Kendall-τ généralisée (*cf.* Définition 1.13 p.17) et de la Kendall-τ induite (*cf.* Définition 2.1 p.40). Comme la première, elle permet de prendre en compte des égalités entre éléments, et comme la seconde elle peut être utilisée avec deux classements incomplets.

Définition 3.1. La mesure de Kendall-τ généralisée induite entre deux classements avec égalités $r, s \in \mathcal{R}_n^I$ est noté $I^{(p)}(r,s)$ avec $p \in]0;1]$ et est définie comme suit :

$$\begin{aligned} I^{(p)}(r,s) = |\{(i,j) \in elem(r) \cap elem(s) : i < j \wedge \\ (r[i] < r[j] \wedge s[i] > s[j] \vee r[i] > r[j] \wedge s[i] < s[j])\}| \\ +p * |\{(i,j) \in elem(r) \cap elem(s) : i < j \wedge \\ (r[i] \neq r[j] \wedge s[i] = s[j] \vee r[i] = r[j] \wedge s[i] \neq s[j])\}| \end{aligned}$$

On rappelle que $elem(r)$ retourne l'ensemble des éléments présents dans le classement incomplet r tel que défini en Section 1.5.1.

Le score de Kemeny généralisé induit est alors la somme des mesures de Kendall-τ généralisée induite :

Définition 3.2. Pour tout classement avec égalités $r \in \mathcal{R}_n^I$ et un ensemble de classements $\mathcal{R} \subseteq \mathcal{R}_n^I$, le score de Kemeny de r par rapport à $\mathcal{R}$ est défini en fonction de $p \in]0;1]$ tel que :

$$K_I^{(p)}(r,\mathcal{R}) = \sum_{s \in \mathcal{R}} I^{(p)}(r,s)$$

Définition 3.3. Le consensus optimal r^* d'un ensemble de classements *incomplets* $\mathcal{R} \subseteq \mathcal{R}_n^I$ est un classement contenant l'ensemble des éléments de $[n]$ ($r^* \in \mathcal{R}_n$) et est défini de la façon suivante :

$$\forall r \in \mathcal{R}_n : K_I^{(p)}(r^*, \mathcal{R}) \leq K_I^{(p)}(r, \mathcal{R})$$

Exemple 3.1. Soit $\mathcal{R}$ un jeu de données incomplet et soit r_I^* le consensus optimal au sens du score de Kemeny généralisé induit :

$$\mathcal{R} = \begin{pmatrix} r_1 = [\{A\}, \{D, E\}] \\ r_2 = [\{D, E\}] \\ r_3 = [\{B\}, \{A, D\}, \{C\}] \\ r_4 = [\{A\}, \{E\}] \end{pmatrix} \qquad \begin{array}{l} r_I^* = [\{B\}, \{A\}, \{D, E\}, \{C\}] \\ K_I^{(1)}(r_I^*, \mathcal{R}) = 1 \end{array}$$

On remarque que C n'est classé que dans r_3 et après A, D et B, il apparaît donc naturellement derrière ces derniers dans r_I^*. Les éléments D et E lorsqu'ils apparaissent dans un même classements sont à égalité, ils le sont donc dans le consensus optimal.

3.2.3 Une pseudo-distance pour une interprétation fine des égalités entre éléments

L'utilisation du processus d'unification (*cf.* Section 2.3.2) permet de produire un jeu de données complet en ajoutant à chaque classement dans un groupe dit d'*unification* les éléments absents de ce classement et présents dans les autres classements du jeu de données original. L'utilisation de jeux de données unifiés avec le score de Kemeny généralisé amène cependant à considérer que les éléments présents dans un groupe d'unification sont à égalité, or nous avons remarqué dans [Bra+14] que cette égalité est artificielle car on ignore justement si ces éléments initialement absents avaient la même (non-)pertinence. L'impact négatif de ces égalités artificielles est illustré par l'Exemple 3.2 où deux éléments qui ne sont initialement jamais à égalité, le sont dans le consensus optimal du fait de l'utilisation conjointe du processus d'unification et de la distance de Kendall-τ généralisée.

Exemple 3.2. Soit $\mathcal{R}_{Unif}$ le jeu de données produit après unification du jeu de données $\mathcal{R}$ (*cf.* Exemple 3.1) et r_G^* est alors le consensus optimal de $\mathcal{R}_{Unif}$ au

sens du score de Kemeny généralisé.

$$\mathcal{R}_{Unif} = \begin{pmatrix} r_1 = [\{A\}, \{D, E\}, \{B, C\}] \\ r_2 = [\{D, E\}, \{A, B, C\}] \\ r_3 = [\{B\}, \{A, D\}, \{C\}, \{E\}] \\ r_4 = [\{A\}, \{D\}, \{B, C, E\}] \end{pmatrix} \quad r^*_G = [\{A\}, \{D, E\}, \{B, C\}]$$

On remarque que dans r^*_G les éléments D et E sont à égalité comme c'est majoritairement le cas dans $\mathcal{R}_{Unif}$. Les éléments B et C sont eux aussi mis à égalité dans r^*_G or ils ne sont jamais à égalité dans le jeu de données original, c'est l'unification qui fait que B et C sont à égalité à trois reprises dans $\mathcal{R}_{Unif}$.

Étendons le processus d'unification afin d'indiquer explicitement quel groupes sont les groupes d'*unification*, c'est-à-dire les groupes nouvellement créés à la fin des classements pour contenir les éléments manquants. Dans l'Exemple 3.3, l'étiquetage de chaque groupe d'unification est indiqué par un u en indice : $\{B, C\}_u$.

Formalisons maintenant la pseudo-distance de Kendall-τ généralisée.

Définition 3.4. Soit r et s deux classements avec égalités de $\mathcal{R}_n$. Soit $unif(r)$ le groupe d'unification de r, on considère que $unif(r) = \emptyset$ lorsque r n'a pas de groupe d'unification. La pseudo-distance de Kendall-τ généralisée adaptée à l'unification est noté $G^{(p)}_U(r, s)$ avec $p \in]0; 1]$ et s'écrit de la façon suivante :

$$\begin{aligned} G^{(p)}_U(r, s) = |\{(i, j) : i < j \wedge (r[i] < r[j] \wedge s[i] > s[j] \vee r[i] > r[j] \wedge s[i] < s[j])\}| \\ + p * |\{(i, j) : i < j \wedge (r[i] \neq r[j] \wedge s[i] = s[j] \wedge i \notin unif(s) \\ \vee r[i] = r[j] \wedge s[i] \neq s[j] \wedge i \notin unif(r))\}| \end{aligned}$$

On remarque que cette mesure n'est pas une métrique car elle ne permet pas de distinguer deux classements différents : $G^{(p)}_U([\{A\}, \{B\}], [\{A, B\}_u]) = 0$. Cependant, c'est une pseudo-métrique (*cf.* Définition 1.3 p.13) car les contraintes de symétrie et d'inégalité triangulaire sont respectées.

L'extension du score de Kemeny généralisé pour prendre en compte le processus d'unification se fait comme étant la somme des distances de Kendall-τ généralisées adaptée à l'unification :

$$K^{(p)}_U(r, \mathcal{R}) = \sum_{s \in \mathcal{R}} G^{(p)}(r, s)$$

Exemple 3.3. Soit $\mathcal{R}_{Unif}$ le jeu de données produit après unification du jeu de données $\mathcal{R}$ (*cf.* Exemple 3.1) et r_F^* est alors le consensus optimal de $\mathcal{R}_{Unif}$ au sens du score de Kemeny généralisé prenant en compte le processus d'unification.

$$\mathcal{R}_{Unif} = \begin{pmatrix} r_1 = [\{A\}, \{D, E\}, \{B, C\}_u] \\ r_2 = [\{D, E\}, \{A, B, C\}_u] \\ r_3 = [\{B\}, \{A, D\}, \{C\}, \{E\}_u] \\ r_4 = [\{A\}, \{D\}, \{B, C, E\}_u] \end{pmatrix} \qquad r_F^* = [\{A\}, \{D, E\}, \{B\}, \{C\}]$$

On remarque une différence entre r_F^* et r_G^* le consensus de l'Exemple 3.2. Alors que les éléments B et C sont à égalité dans r_G^*, ils sont séparés dans r_F^*, tout comme dans le jeu de données original, et cela grâce à la l'utilisation de la pseudo-distance de Kendall-τ.

3.2.4 Conclusion

Nous avons présenté dans cette section deux processus de normalisation et cinq distances et mesures pour prendre en compte des jeux de données incomplets en interprétant différemment l'absence d'un élément. Nous avons présenté un guide à l'utilisateur pour lui permettre de choisir au mieux entre les processus de normalisations, distances et mesures existantes et nouvelles introduites dans ce chapitre. Les approches ont été formalisés, et les exemples ont pu montrer que le choix de la distance et de la normalisation sur le consensus calculé avait un impact non négligeable, justifiant ainsi la nécessité d'un guide à l'utilisateur.

L'extension du score de Kemeny, qui permet d'interpréter plus finement les égalités entre éléments (*cf.* Section 3.2.3), a été créé et formalisé dans ConQuR-Bio [Bra+14]. L'approche complète sera présentée en Section 5.2.

La mesure de Kendall-τ généralisée induite (*cf.* Section 3.2.2) a été conçue pour nous permettre dans [Sta+14] de faire un consensus entre experts avec un droit de réserve. Cette approche sera présentée en Section 5.3.

Nous avons vu au Chapitre 2 que très peu d'algorithmes sont conçus pour prendre en compte les égalités, une problématique qui apparaît pourtant souvent et d'autant plus que le processus d'unification est régulièrement utilisé pour travailler avec des jeux de données incomplets. Les cas d'utilisation menant à utiliser la distance de Kendall-τ induite et sa généralisation étant beaucoup moins fréquents que ceux menant à prendre en compte des égalités, nous nous concentrons dans la section suivante sur l'évolution des algorithmes pour prendre en compte les égalités entre éléments.

3.3 Impact des égalités sur les algorithmes

Nous présentons dans un premier temps une méthodologie générale pour adapter les algorithmes existants pour qu'ils gèrent les égalités. Les algorithmes décrits en Section 2.2.1 gèrent nativement les égalités et n'ont donc pas besoin d'être adaptés. Après avoir introduit une méthodologie générale pour adapter des algorithmes aux égalités, nous les adaptons et détaillons les différents algorithmes en regardant dans un premier temps ceux basés sur la distance de Kendall-τ, puis les algorithmes positionnels.

3.3.1 Méthodologie pour adapter les algorithmes aux égalités

Les algorithmes qui utilisent la distance de Kendall-τ (*cf.* Section 2.2.2) se concentrent sur les désaccords, plus précisément pour une paire d'éléments (a, b) ils comptabilisent le nombre de classements classant a avant b, et réciproquement b avant a. Leurs adaptations pour produire des classements avec égalités reposent donc sur la prise en compte du nombre de classements, où a est à égalité avec b. Trois stratégies générales peuvent être esquissées pour adapter les algorithmes. Tout d'abord, pour les algorithmes fonctionnant avec une construction récursive dont le choix de branchement est basé sur le fait de placer un(des) élément(s) avant ou après un autre, la présence d'égalité apporte un troisième choix : celui de mettre le(s) élément(s) dans le même groupe. Cela peut alors soit entraîner une adaptation de l'algorithme original comme c'est le cas pour KwikSort, soit déboucher sur un nouvel algorithme B&B. Ensuite, pour des algorithmes basés sur de la recherche locale (par exemple, Chanas), de nouvelles opérations doivent être conçues pour explorer l'espace de recherche, et peuvent entraîner la création d'un nouvel algorithme tel que BioConsert. Finalement, pour les algorithmes de programmation linéaire, un nouveau formalisme doit être établi (*cf.* Section 3.4).

Les algorithmes utilisant la position des éléments pour calculer un consensus peuvent être utilisés directement avec le classement avec égalités. La formulation de la position d'un élément correspondant au nombre d'éléments le précédant incrémenté de un, cette formulation couvre autant les permutations que les classements à égalités. Prendre en compte la présence d'égalités entre éléments et le fait de créer/rompre des égalités entre éléments n'est pas directement possible dans ces algorithmes. En effet, l'information de base se concentre non pas sur une paire d'éléments comme pour la distance de Kendall-τ mais sur un élément et sa position. On note cependant que des solutions ad hoc peuvent être conçues.

3.3.2 Adaptation des approches basés sur Kendall-τ

Les algorithmes basés sur la distance de Kendall-τ gèrent les désaccords relatifs à l'inversion d'ordre entre deux éléments, mais ne gèrent pas ceux relatifs aux égalités. Nous étudions ci-après le moyen pour qu'ils gèrent cette deuxième catégorie de désaccords.

3.3.2.1 Algorithmes de recherche locale :

Les algorithmes Chanas [CK96] et ChanasBoth [CW09 ; SZ09] sont construits autour d'un mouvement qui permute des éléments. Adapter ces deux algorithmes similairement à leur fonctionnement actuel nécessiterait donc d'introduire un mouvement pour créer et rompre des égalités entre éléments. Plusieurs mouvements différents permettent de créer et rompre des égalités, il faudrait donc étudier ceux qui fonctionnent le mieux, si cela dépend de la nature des données, etc. Il existe cependant un algorithme (BioConsert [CBDH11]) qui travaille nativement avec des égalités, et gère le coût lié à la création/retrait d'égalité. Cet algorithme peut permuter des éléments, mais il peut surtout déplacer un élément pour créer ou rompre une égalité avec un groupe d'éléments. Enfin cet algorithme a déjà été étudié et publié, l'utiliser permet donc de tirer profit de cette expérience pour avoir un algorithme de recherche locale.

3.3.2.2 Diviser-pour-régner

La formulation de KwikSort [ACN08] peut être adaptée pour inclure les égalités. Les éléments doivent avoir la possibilité d'être mis dans un groupe à égalité avec pivot, en plus d'être placés avant ou après le (groupe du) pivot. On peut de plus prendre en compte les coûts liés à la création/suppression d'égalité entre paires d'éléments. La complexité est modifiée par un facteur constant seulement. L'adaptation de KwikSort est détaillée en Algorithme 1. On note que l'opérateur $\_.\_$ permet de concaténer des classements et des groupes. Considérons deux classements $r = [\mathcal{G}_1^r, .., \mathcal{G}_k^r]$ et $s = [\mathcal{G}_1^s, .., \mathcal{G}_l^s]$ ainsi qu'un groupe d'éléments $\mathcal{G}$, on a alors $r.\mathcal{G} = [\mathcal{G}_1^r, .., \mathcal{G}_k^r, \mathcal{G}]$ et $r.\mathcal{G}.s = (r.\mathcal{G}).s = [\mathcal{G}_1^r, .., \mathcal{G}_k^r, \mathcal{G}, \mathcal{G}_1^s, .., \mathcal{G}_l^s]$

3.3.2.3 Branch-and-bound

L'algorithme B&B [AM12] a été conçu pour prendre en entrée uniquement des permutations. Nous considérons deux variantes de cet algorithme. La première variante permet de calculer une solution optimale au problème de l'agrégation de classements, nous proposons à cette fin une formulation en programmation linéaire pour trouver un consensus optimal dans le cas de l'agrégation de classements avec égalités. Une seconde variante de B&B proposée par ses auteurs permet, grâce à

Algorithme 1 KwikSort ($\mathcal{R} \subseteq \mathcal{R}_n$, $elements \subseteq [n]$)

Si $|elements| \leq 1$ **Alors**
  Retourner $elements$.
FinSi
Choisir aléatoirement i un pivot dans $elements$
$avec \leftarrow \{i\}$, $avant \leftarrow \emptyset$, $après \leftarrow \emptyset$
Pour-Tout $j \in elements \backslash \{i\}$ **Faire**
  $av \leftarrow \sum_{r \in \mathcal{R}} [r[i] < r[j]]$
  $ap \leftarrow \sum_{r \in \mathcal{R}} [r[i] > r[j]]$
  $eg \leftarrow \sum_{r \in \mathcal{R}} [r[i] = r[j]]$
  Si $av > ap$ **et** $av > eg$ **Alors**
    $avant \leftarrow avant \cup \{j\}$
  Sinon
    Si $eg > ap$ **Alors**
      $égale \leftarrow égale \cup \{j\}$
    Sinon
      $après \leftarrow après \cup \{j\}$
    FinSi
  FinSi
Fin pour
Retourner KwikSort ($\mathcal{R}$,$avant$) . $égale$. KwikSort ($\mathcal{R}$,$après$)

l'utilisation de recherche par faisceaux, de calculer des consensus (non nécessairement optimaux). Cependant calculer un consensus de classements avec égalités en suivant l'approche de B&B implique de concevoir une nouvelle construction arborescente de la solution, mais aussi étudier la mise en œuvre de la recherche par faisceaux. L'ensemble de ces tâches relève donc non plus de l'adaptation d'un algorithme, mais de la conception d'un nouvel algorithme.

3.3.2.4 Programmation linéaire

Un nouvel algorithme est introduit en Section 3.4 afin de calculer une solution optimale au problème de l'agrégation de classements. L'approche Ailon$\frac{3}{2}$ [Ail10] qui relaxe le programme linéaire [AM12 ; CDK06 ; SZ09] depuis des nombres entiers vers des nombres réels est utilisée telle qu'elle a été proposée par ses auteurs.

3.3.2.5 Autres

Pick-a-Perm [ACN08] est utilisable directement avec des classements avec égalités dans sa version aléatoire. Dans sa version déterministe, il suffit de calculer lequel des classements à la plus petite distance avec la distance de Kendall-τ généralisée en lieu et place de la distance de Kendall-τ.

RepeatChoice [Ail10] prend en entrée des classements avec égalités et, dans le but de minimiser les désaccords, produit une permutation en fragmentant les groupes selon l'ordre dans lequel on peut trouver les éléments dans un autre classement. Lorsque des égalités persistent dans la solution calculée, elles sont aléatoirement cassées. Le retrait de cette dernière étape permet de produire des classements en sortie.

3.3.3 Adaptation des algorithmes positionnels

3.3.3.1 Système de vote :

La formulation de la position d'un élément (à savoir le nombre d'éléments le précédent plus un) est valable à la fois pour les permutations et les classements. Par conséquent, les algorithmes BordaCount [Bor81], et CopelandMethod [Cop51] peuvent être utilisés avec des classements à égalités.

$$Score_{\text{BordaCount}}(x) = \sum_{r \in \mathcal{R}} r[x]$$

$$Score_{\text{CopelandMethod}}(x) = -\sum_{r \in \mathcal{R}} après(r, x)$$

On rappelle que la fonction $après(r, x)$ retourne le nombre d'éléments suivants x dans r (*cf.* Définition 1.22 p.23).

Pour chaque algorithme, la production d'un consensus se fait alors en classant les éléments par ordre de score croissant. Dans le cas où deux éléments présentent le même score, ils sont retournés dans le même groupe.

Bien que ces algorithmes puissent produire des classements avec égalités, ils ne sont pas en mesure de considérer le coût de créer/rompre ces égalités entre éléments (*cf.* Exemple 3.4).

Exemple 3.4. Soit un jeu de données $\mathcal{R}$ contenant des égalités. Les éléments B et C sont à égalité dans la majorité des classements, et dans le consensus optimal r^*. Cependant la somme des rangs des éléments B et C calculée par BordaCount sera différente pour les deux éléments (13 pour B et 14 pour C), et ils seront alors

séparés dans r_B le consensus calculé par BordaCount.

$$\mathcal{R} = \begin{pmatrix} r_1 = [\{A\}, \{\mathbf{B}, \mathbf{C}\}, \{D\}, \{E\}] \\ r_2 = [\{A\}, \{\mathbf{B}, \mathbf{C}\}, \{E\}, \{D\}] \\ r_3 = [\{D\}, \{A\}, \{\mathbf{B}, \mathbf{C}\}, \{E\}] \\ r_4 = [\{A\}, \{D\}, \{\mathbf{B}, \mathbf{C}\}, \{E\}] \\ r_5 = [\{A\}, \{D\}, \{\mathbf{B}\}, \{\mathbf{C}\}, \{E\}] \end{pmatrix} \quad \begin{array}{l} r^* = [\{A\}, \{D\}, \{\mathbf{B}, \mathbf{C}\}, \{E\}] \\ r_B = [\{A\}, \{D\}, \{\mathbf{B}\}, \{\mathbf{C}\}, \{E\}] \end{array}$$

3.3.3.2 Top-k

MEDRank [FKS03b] est facilement adaptable pour considérer les égalités : dans un classement, plusieurs éléments peuvent être lus en même temps s'ils sont à égalité dans un même groupe. MEDRank est décrit en Algorithme 2 dans une formulation qui est équivalent à celle de [FKS03b] pour les permutations, mais qui permet aussi de considérer les égalités.

Algorithme 2 MEDRank($\mathcal{R} \subseteq \mathcal{R}_n$, h un seuil)

$g \leftarrow \emptyset$, $compteur \leftarrow [0, ..., 0]$ // compteur est un tableau de n entiers
Pour i **de** 1 **à** n **Faire**
  Pour-Tout Classement $r \in \mathcal{R}$ **Faire**
    Pour-Tout Element $k \in r[i]$ **Faire** // $[0; n]$ *éléments pour chaque* $r[i]$.
      $compteur[k] \leftarrow compteur[k] + 1$
      Si $compteur[k] \in [h \times m; m]$ **Alors**
        $compteur[k] \leftarrow m + 1$
        $g \leftarrow g \cup \{k\}$
      FinSi
    Fin pour
  Fin pour
  Si $|g| > 0$ **Alors**
    $c \leftarrow c \,.\, g$
    $g \leftarrow \emptyset$
  FinSi
Fin pour
Retourner c.

3.3.3.3 Hybride

L'approche MC4 [Dwo+01] utilise une représentation à base de graphes et peut prendre les classements avec des égalités en entrée car elle modélise l'ordre

des éléments avec des arcs. Cependant, cette approche est coûteuse, et la prise en compte du coût de créer/rompre des égalités impliquerait non pas une adaptation de la chaîne de Markov existante, mais une toute nouvelle modélisation.

Nous avons, dans cette section, proposé des adaptations des heuristiques et algorithmes d'approximation pour le calcul de consensus au cas de consensus avec égalités. La section suivante propose l'adaptation de l'algorithme exact aux égalités.

3.4 Un nouvel algorithme en PNE pour prendre en compte les égalités

Nous considérons maintenant le problème de la conception d'un algorithme exact capable de prendre en compte les égalités. Le but de la conception d'un tel algorithme est double : calculer des consensus optimaux et connaître à quelle distance se trouve un consensus optimal d'un jeu de données. Ce second but fournira ainsi un point de comparaison (une valeur de distance) à laquelle comparer chaque consensus calculé par une heuristique. Cet algorithme a été proposé par Guillaume Blin dans [Bra+15] et étend la méthode existante de [CDK06] précédemment introduite (*cf.* Section 2.2.2.1 p.29).

Lorsque l'on considère le problème de l'agrégation de classements sans égalité entre éléments, la solution calculée par PNE doit (a) pour chaque paire d'éléments établir un ordre strict entre deux éléments, et (b) respecter la transitivité. L'idée directrice pour étendre le programme linéaire aux égalités est d'ajouter un troisième type de contrainte : la transitivité pour l'égalité entre éléments, c'est-à-dire que dire que si a et b sont à égalité, et b et c aussi alors nécessairement a est à égalité avec c.

Nous formalisons dans un premier temps le programme linéaire permettant de calculer des consensus optimaux pour le problème de l'agrégation de classements, puis dans un deuxième temps nous prouvons qu'il est correct et complet pour calculer un consensus entre classements avec égalités.

Instance du problème de l'agrégation de classements. On considère un jeu de données $\mathcal{R} \in \mathcal{R}_n$ contenant m classements sur les éléments de $[n]$.

Variable. Pour chaque paire d'éléments (a, b), on définit une variable booléenne $x_{a<b}$ afin d'exprimer que dans le consensus optimal a est avant b, la variable prenant alors comme valeur 1 lorsque a précède b. Deux éléments pouvant être à égalité dans

le consensus optimal, on introduit une nouvelle variable $x_{a=b}$, elle aussi booléenne et égale à 1 lorsque les deux éléments sont à égalité.

$$\forall a, b \in [n] : x_{a<b} \in \{0, 1\}$$

$$\forall a, b \in [n], a < b : x_{a=b} \in \{0, 1\}$$

On définit $w_{a<b}$ afin de compter le nombre de classements dans lesquels a est effectivement avant b.

$$\forall a, b \in [n] : w_{a<b} = \sum_{r \in \mathcal{R}} [r[a] < r[b]]$$

Similairement on définit $w_{a=b}$ afin de compter le nombre de classements dans lesquels a et b sont à égalité.

$$\forall a, b \in [n], a < b : w_{a=b} = \sum_{r \in \mathcal{R}} [r[a] = r[b]]$$

Fonction objectif. L'objectif de ce nouveau PNE est d'exprimer la distance de Kendall-τ généralisée en utilisant les variables nouvellement introduites permettant de rendre compte de l'égalité entre éléments. Il s'agira ensuite de minimiser cette fonction objectif.

$$\sum_{\substack{a,b \in [n] \\ a<b}} ((w_{b<a} + w_{a=b})x_{a<b} + (w_{a<b} + w_{a=b})x_{b<a} + (w_{a<b} + w_{b<a})x_{a=b})$$

Contraintes. Par les contraintes, on impose que la solution soit cohérente, c'est-à-dire qu'un élément est soit avant soit à égalité soit après un autre, et cela de façon exclusive. On a donc :

$$\forall a, b \in [n] : x_{a<b} + x_{b<a} + x_{a=b} = 1 \tag{3.1}$$

Afin de conserver la transitivité dans un classement, à savoir que si a est avant b, et que b est avant c alors a est nécessairement avant c, on a :

$$\forall a, b, c \in [n] : x_{a<c} - x_{a<b} - x_{b<c} \geq -1 \tag{3.2}$$

Finalement, la transitivité pour l'égalité, à savoir que si a et b sont à égalité, ainsi que b et c, alors ils sont tous les trois à égalité, on a :

$$\forall a, b, c \in [n] : 2x_{a<b} + 2x_{b<a} + 2x_{b<c} + 2x_{c<b} - x_{a<c} - x_{c<a} \geq 0 \tag{3.3}$$

Lemme 3.1. La traduction en PNE résout correctement le problème de l'agrégation de classements pour trouver un consensus optimal en considérant la distance de Kendall-τ généralisée.

Preuve. Prouvons dans un premier temps que pour tout classement il existe une affectation de valeurs pour les variables de notre PNE décrivant ce classement.

Soit un consensus optimal c^*. Pour chaque paire d'éléments (a, b) soit a est avant b et dans ce cas $x_{a<b} = 1$, soit a est après b et dans ce cas $x_{b<a} = 1$ soit finalement a et b sont à égalité et donc $x_{b=a} = 1$. Ces trois choix étant exclusifs, la contrainte (3.1) est respectée pour toute paire d'éléments (a, b)

Sachant que c^* est un classement avec égalités, la transitivité est nécessairement respectée. Soit un triplet d'éléments (a, b, c) où a est avant b (donc $x_{a<b} = 1$) et b est avant c (donc $x_{b<c} = 1$). La seule façon de ne pas respecter la contrainte (3.2) serait d'avoir $x_{a<c} = 0$, c'est-à-dire que a n'est pas avant c, or c'est impossible car dans un classement avec égalités si a précède b qui lui précède c alors a précède nécessairement c. Si maintenant pour le triplet d'éléments (a, b, c) a n'est pas avant b ou alors si b n'est pas avant c, on a alors la contrainte (3.2) qui est nécessairement respectée.

Finalement, considérons la contrainte (3.3) et un triplet d'éléments (a, b, c). Si au moins deux des éléments ne sont pas à égalité dans le classement, au moins une des variables $x_{a<b}, x_{b<a}, x_{b<c}, x_{c<b}$ est à 1, et donc la contrainte est respectée (pour toute affectation de $x_{a<c}$ et $x_{c<a}$). Supposons maintenant que ces trois éléments sont à égalité, les variables $x_{a<b}, x_{b<a}, x_{b<c}, x_{c<b}$ sont alors à 0. Le seul moyen de ne pas respecter la contrainte (3.3) serait d'avoir $x_{a<c} = 1$ ou $x_{c<a} = 1$, ce qui contredirait l'hypothèse que les trois éléments sont à égalité.

Nous venons de prouver que tout classement peut être représenté par notre PNE.

Prouvons maintenant que toute solution de notre PNE correspond à un classement avec égalités. En considérons une quelconque affectation de variables respectant les contraintes du dit PNE, la contrainte (3.1) assure que pour toute paire d'éléments, ils seront de façon exclusive l'un avant ou après l'autre ou encore a égalité ce qui est le cas avec des classements avec égalités.

Ensuite, la contrainte (3.2) nous assure la transitivité, en effet si $x_{a<b} = 1$ et $x_{b<c} = 1$, alors la contrainte impose $x_{a<c} = 1$ ce qui réalise la transitivité en plaçant a avant c. Si $x_{a<b} \neq 1$ ou $x_{b<c} \neq 1$, alors $x_{a<c}$ prend peut prendre comme valeur 0 ou 1 sans être en contradiction avec la transitivité.

Finalement, la contrainte (3.3) nous assure la transitivité pour l'égalité, à savoir que si les quatre premières variables $(x_{a<b}, x_{b<a}, x_{b<c}, x_{c<b})$ sont à 0 on a

nécessairement aussi $x_{a<c}$ et $x_{c<a}$ à 0 c'est-à-dire que si b est à égalité avec a et c alors nécessairement a et c sont à égalité. Si une des quatre premières variables n'est pas à 0, alors les éléments associés ne sont pas tous à égalité, ce qui est compatible avec un classement à égalité.

Nous venons de prouver que toute solution du PNE est un classement avec égalités.

La fonction objectif correspondant parfaitement à la distance de Kendall-τ généralisée, une solution minimisant de façon optimale la fonction objectif de notre PNE sera alors un consensus optimal dans le cadre de l'agrégation de classements. □

Une toute première implémentation de cette traduction en programmation en nombres entiers a été réalisé par Guillaume Blin en utilisant Java, Python et CPLEX. J'ai par la suite ré-implémenté ce PNE en Java et CPLEX pour obtenir de meilleurs temps de calcul et une implémentation plus modulaire. On nommera par la suite cet algorithme **ExactPNE**.

3.5 Réduction de la complexité de BioConsert

BioConsert est un algorithme glouton de recherche locale introduit par [CBDH11], ainsi qu'au Chapitre 2 (p.26). Son fonctionnement repose sur le déplacement d'un élément au sein d'un classement, et c'est sur ce point que porte l'amélioration. Dans cette section nous proposons de rechercher une position à laquelle déplacer un élément en temps linéaire (en nombre d'éléments) en lieu et place d'un recherche faite en temps au mieux quadratique. On note qu'une implémentation naïve de l'algorithme aboutit à rechercher une position avec une complexité cubique [1].

Dans cette section, nous présentons dans un premier temps l'algorithme de BioConsert tel qu'il a été publié dans [CBDH11]. Dans un deuxième temps, nous présentons les structures de données permettant d'accéder en temps constant à la position d'un élément dans un classement ou encore le nombre de classements plaçant un élément avec un autre. Dans un troisième temps nous présentons l'idée générale qui a permis d'améliorer la recherche d'une modification d'un classement pour réduire sa distance au classement en entrées. Dans un quatrième temps, nous formalisons les fonctions récursives permettant de calculer rapidement la variation

1. Il faut en effet calculer pour chaque position ($\mathcal{O}(n)$) la distance qui, dans une implémentation naïve, est en $\mathcal{O}(n^2)$. Cette recherche naïve aurait alors une complexité en $\mathcal{O}(n^3)$

de distance pour le déplacement d'un élément à une position donnée, et prouvons qu'elles calculent les différences de distance. Dans un cinquième temps nous présentons l'algorithme suivi pour calculer l'ensemble des valeurs des fonctions récursives en temps globalement linéaire. Dans un sixième temps nous présentons l'algorithme de BioConsert mettant en œuvre l'amélioration et discutons sa complexité, pour finalement conclure.

3.5.1 L'algorithme BioConsert dans sa version publiée

BioConsert est décrit en Algorithme 3. On note que l'algorithme publié dans [CBDH11] est une version simplifiée de l'algorithme mis à disposition par les auteurs. Nous reproduisons ici l'algorithme mis à disposition.

BioConsert considère plusieurs classements comme point de départ (ligne 2). Les points de départ sont les classements présents dans $\mathcal{R}$ ainsi qu'un classement où tous les éléments sont à égalité. La modification d'un classement est gloutonne : il est modifié si et seulement si cela réduit la distance de Kendall-τ généralisée entre ce classement et les classements de $\mathcal{R}$. L'algorithme boucle tant qu'il trouve des modifications réduisant la distance. Des classements obtenus depuis les points de départ, le classement obtenu ayant la plus faible distance avec $\mathcal{R}$ est retourné par BioConsert.

L'évaluation d'une position pour y déplacer un élément (ligne 10) se fait en temps linéaire, elle est cependant répétée pour toutes les positions. L'amélioration majeure de l'Algorithme 3, développé dans le cadre de [Bra+15], a été d'évaluer toutes les positions simultanément en temps globalement linéaire. Cette amélioration est présentée ci-après, et son implémentation en Java est disponible en Appendice A.

3.5.2 Structure de données et notations

Dans la suite de cette section nous réutilisons les notations et structures de données utilisées pour ExactPNE dans la section précédente. On rappelle que $w_{a<b}$ représente le nombre de classements où a est avant b, et $w_{a=b}$ le nombre de classements où a et b sont à égalité. On considère que l'accès à ces valeurs se fait en temps constant, tout comme l'accès à $r[i]$, la position d'un élément i dans un classement r.

Algorithme 3 BioConsert2011($\mathcal{R} \subseteq \mathcal{R}_n$, $p \in]0;1]$ paramètre du score de Kemeny)

1: $c \leftarrow [\{1,\ldots,n\}]$
2: **Pour-Tout** $r \in \mathcal{R} \cup \{\ [\{1,\ldots,n\}]\ \}$ **Faire**
3: $encore \leftarrow 1$;
4: **Tant-que** $encore > 0$ **Faire**
5: $encore \leftarrow 0$; $change \leftarrow 0$
6: **Pour** e **de** 1 **à** n **Faire** ..... // $\mathcal{O}(n^3)$
7: $change \leftarrow 0$;
8: $j \leftarrow 0$;
9: **Tant-que** $j \leq nbGroup(r)$ **et** $change = 0$ **Faire** ..... // $\mathcal{O}(n^2)$
10: **Si** $\delta_x(r,e,j) < 0$ **Alors** ..... // $\mathcal{O}(n\ )$
11: $r \leftarrow changeDeGroupe(e,j)(r)$
12: $change \leftarrow 1$
13: **FinSi**
14: $j \leftarrow j+1$
15: **Fin Tant-Que**
16: $j \leftarrow 0$;
17:
18: **Tant-que** $j \leq nbGroup(r)+1$ **et** $change = 0$ **Faire** .... // $\mathcal{O}(n^2)$
19: **Si** $\delta_a(r,e,j) < 0$ **Alors** ..... // $\mathcal{O}(n\ )$
20: $r \leftarrow nouveauGroupe(e,j)(r)$
21: $change \leftarrow 1$
22: **FinSi**
23: $j \leftarrow j+1$
24: **Fin Tant-Que**
25: $encore = encore + change$
26: **Fin pour**
27: **Fin Tant-Que**
28: **Si** $K^{(p)}(r,\mathcal{R}) < K^{(p)}(c,\mathcal{R})$ **Alors**
29: $c \leftarrow r$
30: **FinSi**
31: **Fin pour**
32: **Retourner** c.

3.5.3 Relation de récurrence entre les déplacements d'un élément

L'idée directrice de l'amélioration se base sur une récurrence. Concentrons-nous dans un premier temps sur le déplacement de e dans un nouveau groupe, en nous aidant de la Figure 3.2.

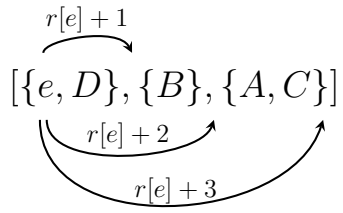

FIGURE 3.2 – Déplacements de e.

Pour **déplacer e** à la position $\mathbf{r[e]+1}$, la variation de distance sera uniquement lié aux éléments en position $r[e]$, anciennement à égalité avec e. Dans notre exemple, c'est l'élément D avec lequel on va rompre l'égalité, et être en désaccords avec les classements plaçant D après e.

Pour **déplacer e** à la position $\mathbf{r[e]+2}$, la variation de distance concerne donc les désaccords liés au déplacement en $r[e]+1$ (désaccords avec l'élément D), plus ceux en rapport avec les éléments anciennement à $r[e]+1$ (inversion d'ordre avec l'élément B).

Pour **déplacer e** à la position $\mathbf{r[e]+3}$, la variation de distance sera donc celle pour le déplacer à $r[e]+2$, plus l'inversion d'ordre avec les éléments A et C.

3.5.4 Formalisation du calcul de la variation de distance

Nous formalisons ci-après le calcul de variation de distance lors du déplacement d'un élément dans un (nouveau) groupe d'éléments sous la forme de deux formules de récurrence, et prouvons qu'elles modélisent la variation de désaccords lors du déplacement d'un élément dans un (nouveau) groupe. Nous fournissons ensuite les algorithmes des fonctions retournant une position dans un classement pour laquelle un élément données peut être déplacé afin que cela réduise la distance entre ce classement et le jeu de données.

3.5.4.1 Déplacement d'un élément dans un nouveau groupe

Nous définissons ci après une fonction retournant la variation de distance si, au sein du classement r, l'on déplace e dans un nouveau groupe **a**jouté à la position t. Nommons cette fonction $\delta_a(r, e, t)$. Nous définissons de plus les fonction $\sigma_a^+(r, e, t)$, $\sigma_a^-(r, e, t)$, $C_a^+(r, e)$, $C_a^-(r, e)$ permettant de décomposer $\delta_a(r, e, t)$ et ainsi améliorer sa lisibilité.

Proposition 3.1. Soit r un classement à égalité, e un élément et $t \in [1; nbGroupe(r) + 1]$ une position pour laquelle on souhaite connaître la variation de distance en déplaçant l'élément e dans un nouveau groupe à la position t.

$$\delta_a(r,e,t) = \begin{cases} t < r[e]: & \delta_a(r,e,t+1) & +\sigma_a^+(r,e,t) \\ t = r[e]: & C_a^+(r,e) & \\ t = r[e]+1: & C_a^-(r,e) & \\ t > r[e]+1: & \delta_a(r,e,t-1) & +\sigma_a^-(r,e,t) \end{cases}$$

$$\sigma_a^+(r,e,t) = \sum_{\substack{i \in N \\ e \neq i \\ r[i]=t}} (w_{i<e} - w_{e<i})$$

$$C_a^+(r,e) = \sum_{\substack{i \in N \\ e \neq i \\ r[i]=r[e]}} w_{i<e} + p \sum_{\substack{i \in N \\ e \neq i \\ r[i]=r[e]}} (w_{i=e} - w_{e<i} - w_{i<e})$$

$$C_a^-(r,e) = \sum_{\substack{i \in N \\ e \neq i \\ r[i]=r[e]}} w_{e<i} + p \sum_{\substack{i \in N \\ e \neq i \\ r[i]=r[e]}} (w_{i=e} - w_{e<i} - w_{i<e})$$

$$\sigma_a^-(r,e,t) = \sum_{\substack{i \in N \\ e \neq i \\ r[i]=t-1}} (w_{e<i} - w_{i<e})$$

Preuve. Montrons que $\delta_a(r,e,j)$ permet de calculer la variation de distance. Pour cela nous montrons qu'un calcul "classique" de la variation de distance pour une position quelconque j est égale à $\delta_a(r,e,j)$.

Soit r un classement de $\mathcal{R}_n$ de longueur k, soit e un élément du classement r, et $j \in [1;k]$ la position pour laquelle on veut connaître la variation de distance si on y déplace l'élément e.

$$r = [\mathcal{G}_1, ..., \mathcal{G}_{j-1}, \mathcal{G}_j, ..., \underset{\mathcal{G}_{r[e]}}{\{e, ...\}}, \mathcal{G}_k]$$

Déplacer l'élément e dans le classement r influera seulement sur les désaccords relatifs à ce dernier, appelons cette variation de désaccords $\Delta_a(r,e,t)$.

Comptabilisons les désaccords relatifs à la présence de e à la position $r[e]$. Ce sont les $w_{e<i}$ pour les éléments i précédant e, les $w_{i<e}$ pour les éléments i suivant

e, les $p * w_{e=i}$ pour les éléments n'étant pas à égalité avec e et $p * (w_{e<i} + w_{i<e})$ pour les éléments à égalité avec e :

$$\sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\in[1;r[e]-1]}} w_{e<i} + \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\in[r[e]+1;k]}} w_{i<e} + p \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\neq r[e]}} w_{e=i} + p \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]=r[e]}} (w_{e<i} + w_{i<e}) \quad (3.4)$$

Similairement, en plaçant e à la position j, c'est-à-dire entre $\mathcal{G}_{j-1}$ et $\mathcal{G}_j$, les désaccords sont ceux d'inversion d'ordre $w_{e<i}$ pour les éléments avant la position j, les désaccords $w_{i<e}$ pour les autres éléments. Finalement, il y a aussi le fait de ne pas être à égalité avec l'ensemble des éléments :

$$\sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\in[1;j-1]}} w_{e<i} + \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\in[j;k]}} w_{i<e} + p \sum_{\substack{i\in[n]\\ e\neq i}} (w_{e=i}) \quad (3.5)$$

Nous décomposons la variation de désaccords Δ_a en deux cas en fonction de la position de j :

$$\Delta_a(r,e,j)) \begin{cases} j \in [1; r[e]] & : \Delta_a^{inf}(r,e,j) \\ j \in [r[e]+1; k+1] & : \Delta_a^{sup}(r,e,j) \end{cases}$$

La variation de désaccords $\Delta_a^{inf}(r,e,j)$ est alors la différence des sommes 3.5 et 3.4 :

$$\begin{aligned}\Delta_a^{inf}(r,e,j) = &- \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\in\\ [1;r[e]-1]}} w_{e<i} - \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\in\\ [r[e]+1;k]}} w_{i<e} - p \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\neq r[e]}} w_{e=i} - p \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]=r[e]}} w_{e<i} + w_{i<e} \\ &+ \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\in[1;j-1]}} w_{e<i} + \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\in[j;k]}} w_{i<e} + p \sum_{\substack{i\in[n]\\ e\neq i}} w_{e=i}\end{aligned}$$

On groupe les sommes portant sur les même terme :

$$\begin{aligned}\Delta_a^{inf}(r,e,j) = &- \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\in[j;r[e]-1]}} w_{e<i} + \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\in[j;r[e]]}} w_{i<e} \\ &+ p \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]=r[e]}} w_{e=i} - p \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]=r[e]}} (w_{e<i} + w_{i<e})\end{aligned}$$

On regroupe les sommes :

$$\begin{aligned}\Delta_a^{inf}(r,e,j) = & \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\in[j;r[e]-1]}} (w_{i<e} - w_{e<i}) + \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]=r[e]}} (w_{i<e}) \\ & +p \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]=r[e]}} (w_{e=i} - w_{e<i} - w_{i<e})\end{aligned} \tag{3.6}$$

Évaluons maintenant $\delta_a(r,e,j)$ pour $j \in [1; r[e]]$. Pour cela, ré-écrivons la fonction de façon non-récursive, puis évaluons cette forme ré-écrite.
Supposons que :

$$\forall j \in [1; r[e]] : \delta_a(r,e,j) = C_a^+(r,e) + \sum_{t=j}^{r[e]-1} \sigma_a^+(r,e,t) \tag{3.7}$$

Si $j = r[e]$ alors $\delta_a(r,e,j) = C_a^+(r,e)$, et donc l'Équation 3.7 est valide

Par récurrence, supposons que l'Équation 3.7 soit vérifié pour $j+1$ et montrons alors qu'elle est vérifié pour j

$$\begin{aligned}\delta_a(r,e,j) =& \delta_a(r,e,j+1) + \sigma_a^+(r,e,j) \\ =& \left(C_a^+(r,e) + \sum_{t=j+1}^{r[e]-1} \sigma_a^+(r,e,t) \right) + \sigma_a^+(r,e,j) \\ =& C_a^+(r,e) + \sum_{t=j}^{r[e]-1} \sigma_a^+(r,e,t)\end{aligned}$$

L'Équation 3.7 est donc vrai $\forall j \in [1; r[e]]$. La fonction $\delta_a(r,e,j)$ peut alors être ré-écrite telle que décrite dans cette équation.

Évaluons maintenant la forme non récursive de $\delta_a(r,e,j)$ pour $j \in [1; r[e]]$.

$$\begin{aligned}
\delta_a(r,e,j) =& C_a^+(r,e) + \sum_{t=j}^{r[e]-1} \sigma_a^+(r,e,t) \\
=& C_a^+(r,e) + \sum_{t=j}^{r[e]-1} \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]=t}} (w_{i<e} - w_{e<i}) \\
=& C_a^+(r,e) + \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\in[j;r[e]-1]}} (w_{i<e} - w_{e<i}) \\
=& \left(\sum_{\substack{i\in N\\ e\neq i\\ r[i]=r[e]}} w_{i<e} + p \sum_{\substack{i\in N\\ e\neq i\\ r[i]=r[e]}} (w_{i=e} - w_{e<i} - w_{i<e}) \right) + \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\in\\ [j;r[e]-1]}} (w_{i<e} - w_{e<i})
\end{aligned}$$

En identifiant terme à terme avec l'Équation 3.6, on a $\delta_a(r,e,j) = \Delta_a^{inf}(r,e,j), \forall j \in [1; r[e]]$.

Étudions maintenant $\Delta_a^{sup}(r,e,t)$, cette dernière étant elle aussi la différence des sommes 3.5 et 3.4. $\Delta_a^{sup}(r,e,t)$ est développée différemment de $\Delta_a^{inf}(r,e,t)$ car $j \in [r[e]+1; k+1]$, et donc les membres des sommes qui s'annule ne sont alors plus les mêmes :

$$\begin{aligned}
\Delta_a^{sup}(r,e,j) = &- \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\in\\ [1;r[e]-1]}} w_{e<i} - \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\in\\ [r[e]+1;k]}} w_{i<e} - p \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\neq r[e]}} w_{e=i} - p \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]=r[e]}} w_{e<i} + w_{i<e} \\
&+ \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\in[1;j-1]}} w_{e<i} + \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\in[j;k]}} w_{i<e} + p \sum_{\substack{i\in[n]\\ e\neq i}} w_{e=i}
\end{aligned}$$

On groupe les sommes portant sur les même terme :

$$\begin{aligned}\Delta_a^{sup}(r,e,j) = & \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\in[r[e];j-1]}} w_{e<i} - \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\in[r[e]+1;j-1]}} w_{i<e} \\ & + p \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]=r[e]}} w_{e=i} - p \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]=r[e]}} (w_{e<i} + w_{i<e})\end{aligned}$$

On regroupe les sommes :

$$\begin{aligned}\Delta_a^{sup}(r,e,j) = & \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\in[r[e]+1;j-1]}} (w_{e<i} - w_{i<e}) + \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]=r[e]}} w_{e<i} \\ & + p \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]=r[e]}} (w_{e=i} - w_{e<i} - w_{i<e})\end{aligned} \tag{3.8}$$

Évaluons maintenant $\delta_a(r,e,j)$ pour $j \in [r[e]+1;k]$. Pour cela, ré-écrivons la fonction de façon non-récursive, puis évaluons cette forme ré-écrite.
Supposons que :

$$\forall j \in [r[e]+1;k] : \delta_a(r,e,j) = C_a^-(r,e) + \sum_{t=r[e]+2}^{j} \sigma_a^-(r,e,t) \tag{3.9}$$

Si $j = r[e]+1$ alors $\delta_a(r,e,j) = C_a^-(r,e)$, et donc l'Équation 3.9 est valide
Par récurrence, supposons que l'Équation 3.9 soit vérifié pour $j-1$ et montrons alors qu'elle est vérifié pour j

$$\begin{aligned}\delta_a(r,e,j) =& \delta_a(r,e,j-1) + \sigma_a^-(r,e,j) \\ =& \left(C_a^-(r,e) + \sum_{t=r[e]+2}^{j-1} \sigma_a^-(r,e,t) \right) + \sigma_a^-(r,e,j) \\ =& C_a^-(r,e) + \sum_{t=r[e]+2}^{j} \sigma_a^-(r,e,t)\end{aligned}$$

L'Équation 3.9 est donc vrai $\forall j \in [r[e]+1;k]$. La fonction $\delta_a(r,e,j)$ peut alors être ré-écrite telle que décrite dans cette équation.

Évaluons maintenant la forme non récursive de $\delta_a(r, e, j)$ pour $j \in [r[e]+1; k]$.

$$\begin{aligned}
\delta_a(r,e,j) &= C_a^-(r,e) + \sum_{t=r[e]+2}^{j} \sigma_a^-(r,e,t) \\
&= C_a^-(r,e) + \sum_{t=r[e]+2}^{j} \sum_{\substack{i\in N \\ e\neq i \\ r[i]=t-1}} w_{e<i} - w_{i<e} \\
&= C_a^-(r,e) + \sum_{\substack{i\in[n] \\ e\neq i \\ r[i]\in[r[e]+1;j-1]}} w_{e<i} - w_{i<e} \\
&= \left(\sum_{\substack{i\in N \\ e\neq i \\ r[i]=r[e]}} w_{e<i} + p \sum_{\substack{i\in N \\ e\neq i \\ r[i]=r[e]}} (w_{i=e} - w_{e<i} - w_{i<e}) \right) + \sum_{\substack{i\in[n] \\ e\neq i \\ r[i]\in \\ [r[e]+1;j-1]}} (w_{e<i} - w_{i<e})
\end{aligned}$$

En identifiant terme à terme avec l'Équation 3.8, on a donc $\delta_a(r, e, j) = \Delta_a^{sup}(r, e, j)$ pour $j \in [r[e] + 1; k + 1]$.

On conclut alors que $\forall j \in [1; k + 1]$, $\delta_a(r, e, j) = \Delta_a(r, e, j)$, et donc que $\delta_a(r, e, j)$ permet de décrire la variation de désaccords lors du déplacement de e à la position j. □

3.5.4.2 Déplacement d'un élément dans un groupe existant

La fonction qui permet de retourner la variation de distance lorsque l'on déplace un élément e dans un groupe d'éléments à égalité déjà e<u>x</u>istant à la position t se nomme $\delta_x(r, e, t)$. Elle se décompose en deux sous-fonctions dépendant de si l'on évalue une destination pour e avant ou après sa position actuelle $r[e]$. Nous proposons en Proposition 3.2 une écriture pour cette fonction et prouvons qu'elle permet de retourner la différence de distance en déplaçant l'élément e à la position t dans le classement r.

Proposition 3.2. Soit r un classement à égalité, e un élément et $t \in [1; nbGroupe(r)]$ une position pour laquelle on souhaite connaître la variation

de distance en déplaçant l'élément e dans un groupe existant à la position t.

$$\delta_x(r,e,t) = \begin{cases} t < r[e]: & \delta_x(r,e,t+1) + \sigma_x^+(r,e,t) \\ t = r[e]: & 0 \\ t > r[e]: & \delta_x(r,e,t-1) + \sigma_x^-(r,e,t) \end{cases}$$

$$\begin{aligned} \sigma_x^+(r,e,t) = \quad & +p \sum_{\substack{i\in N \\ e\neq i \\ r[i]=t+1}} (w_{i=e} - w_{i<e} - w_{e<i}) \quad + \sum_{\substack{i\in N \\ e\neq i \\ r[i]=t+1}} w_{i<e} \\ & +p \sum_{\substack{i\in N \\ e\neq i \\ r[i]=t}} (-w_{i=e} + w_{e<i} + w_{i<e}) \quad + \sum_{\substack{i\in N \\ e\neq i \\ r[i]=t}} -w_{e<i} \end{aligned}$$

$$\begin{aligned} \sigma_x^-(r,e,t) = \quad & +p \sum_{\substack{i\in N \\ e\neq i \\ r[i]=t-1}} (w_{i=e} - w_{e<i} - w_{i<e}) \quad + \sum_{\substack{i\in N \\ e\neq i \\ r[i]=t-1}} w_{e<i} \\ & +p \sum_{\substack{i\in N \\ e\neq i \\ r[i]=t}} (-w_{i=e} + w_{e<i} + w_{i<e}) \quad + \sum_{\substack{i\in N \\ e\neq i \\ r[i]=t}} -w_{i<e} \end{aligned}$$

Preuve. Montrons que $\delta_x(r,e,j)$ permet de calculer la variation de distance lors du déplacement d'un élément dans un groupe existant. Pour cela nous montrons qu'un calcul "classique" de la variation de distance pour une position quelconque j est égale à $\delta_x(r,e,j)$.

Soit r un classement de $\mathcal{R}_n$ de longueur k, soit e un élément du classement r, et $j \in [1;k]$ la position pour laquelle on veut connaître la variation de distance si on y déplace l'élément e.

$$r = [\mathcal{G}_1, ..., \mathcal{G}_{j-1}, \mathcal{G}_j, ..., \underset{\mathcal{G}_{r[e]}}{\{e, ...\}}, \mathcal{G}_k]$$

Déplacer l'élément e dans le classement r influera seulement sur les désaccords relatifs à ce dernier, appelons cette variation de désaccords $\Delta_x(r,e,t)$.

Comptabilisons les désaccords relatifs à la présence de e à la position $r[e]$. Ce sont les $w_{e<i}$ pour les éléments i précédant e, les $w_{i<e}$ pour les éléments i suivant e, les $p * w_{e=i}$ pour les éléments n'étant pas à égalité avec e et $p * (w_{e<i} + w_{i<e})$ pour les éléments à égalité avec e :

$$\sum_{\substack{i\in[n] \\ e\neq i \\ r[i]\in[1;r[e]-1]}} w_{e<i} + \sum_{\substack{i\in[n] \\ e\neq i \\ r[i]\in[r[e]+1;k]}} w_{i<e} + p \sum_{\substack{i\in[n] \\ e\neq i \\ r[i]\neq r[e]}} w_{e=i} + p \sum_{\substack{i\in[n] \\ e\neq i \\ r[i]=r[e]}} (w_{e<i} + w_{i<e}) \quad (3.10)$$

Similairement, en plaçant e à la position j les désaccords sont :

$$\sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\in[1;j-1]}} w_{e<i} + \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\in[j+1;k]}} w_{i<e} + p\sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\neq j}} w_{e=i} + p\sum_{\substack{i\in[n]\\ e\neq i\\ r[i]=j}} (w_{e<i}+w_{i<e}) \tag{3.11}$$

Nous décomposons la variation de désaccords Δ_x en trois cas en fonction de la position de j :

$$\Delta_x(r,e,j)) \begin{cases} j \in [1; r[e]-1] & : \Delta_x^{inf}(r,e,j) \\ j = r[e] & : 0 \\ j \in [r[e]+1; k] & : \Delta_x^{sup}(r,e,j) \end{cases}$$

La variation de désaccords $\Delta_x^{inf}(r,e,j)$ est alors les désaccords créés en plaçant e dans le groupe j (somme 3.11) auxquels on retire les désaccords liés à la présence de e dans son groupe d'origine (somme 3.10) :

$$\begin{aligned}\Delta_x^{inf}(r,e,j) = &\sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\in[1;j-1]}} w_{e<i} + \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\in[j+1;k]}} w_{i<e} + p\sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\neq j}} w_{e=i} + p\sum_{\substack{i\in[n]\\ e\neq i\\ r[i]=j}} (w_{e<i}+w_{i<e}) \\ &- \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\in\\ [1;r[e]-1]}} w_{e<i} - \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\in\\ [r[e]+1;k]}} w_{i<e} - p\sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\neq r[e]}} w_{e=i} - p\sum_{\substack{i\in[n]\\ e\neq i\\ r[i]=r[e]}} (w_{e<i}+w_{i<e})\end{aligned}$$

On groupe les sommes portant sur les $w_{e<i}$ et $w_{i<e}$:

$$\begin{aligned}\Delta_x^{inf}(r,e,j) = &-\sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\in[j;r[e]-1]}} w_{e<i} + p\sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\neq j}} w_{e=i} + p\sum_{\substack{i\in[n]\\ e\neq i\\ r[i]=j}} (w_{e<i}+w_{i<e}) \\ &\sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\in[j+1;r[e]]}} w_{i<e} - p\sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\neq r[e]}} w_{e=i} - p\sum_{\substack{i\in[n]\\ e\neq i\\ r[i]=r[e]}} (w_{e<i}+w_{i<e})\end{aligned}$$

On groupe les sommes portant sur les $w_{e=i}$, en effet ces sommes s'annulent pour

toutes valeurs différentes de $r[e]$ ou j :

$$\Delta_x^{inf}(r,e,j) = - \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\in[j;r[e]-1]}} w_{e<i} \quad + p \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]=r[e]}} w_{e=i} + p \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]=j}} (w_{e<i} + w_{i<e})$$
$$\sum_{\substack{i\in[n]\\ e\neq i\\ r[i]\in[j+1;r[e]]}} w_{i<e} \quad - p \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]=j}} w_{e=i} - p \sum_{\substack{i\in[n]\\ e\neq i\\ r[i]=r[e]}} (w_{e<i} + w_{i<e})$$

Évaluons maintenant $\delta_x(r,e,j)$ pour $j \in [1; r[e]-1]$ en la ré-écrivant de façon non récursive. Pour cela, ré-écrivons la fonction de façon non-récursive, puis évaluons cette forme ré-écrite.
Supposons que :

$$\forall j \in [1; r[e]-1] : \delta_x(r,e,j) = \sum_{t=j}^{r[e]-1} \sigma_x^+(r,e,t) \tag{3.12}$$

Si $j = r[e]-1$ alors $\delta_x(r,e,j) = C_x^+(r,e)$, et donc l'Équation 3.12 est valide

Par récurrence, supposons que l'Équation 3.12 soit vérifié pour $j+1$ et montrons alors qu'elle est vérifié pour j

$$\begin{aligned}\delta_x(r,e,j) =& \delta_x(r,e,j+1) + \sigma_x^+(r,e,j)\\ =& \left(\sum_{t=j+1}^{r[e]-1} \sigma_x^+(r,e,t)\right) + \sigma_x^+(r,e,j)\\ =& \sum_{t=j}^{r[e]-1} \sigma_x^+(r,e,t)\end{aligned}$$

L'Équation 3.12 est donc vrai $\forall j \in [r[e]+1; k]$. La fonction $\delta_x(r,e,j)$ peut alors être ré-écrite telle que décrite dans cette équation.

Évaluons maintenant la forme non récursive de $\delta_x(r,e,j)$ pour $j \in [r[e]+1;k]$.

$$\begin{aligned}
\delta_x(r,e,j) &= \sum_{t=j}^{r[e]-1} \sigma_x^+(r,e,t) \\
&= \sum_{t=j}^{r[e]-1} \begin{pmatrix} +p \sum\limits_{\substack{i\in N\\ e\neq i\\ r[i]=t+1}} (w_{i=e} - w_{i<e} - w_{e<i}) & + \sum\limits_{\substack{i\in N\\ e\neq i\\ r[i]=t+1}} w_{i<e} \\ +p \sum\limits_{\substack{i\in N\\ e\neq i\\ r[i]=t}} (-w_{i=e} + w_{e<i} + w_{i<e}) & + \sum\limits_{\substack{i\in N\\ e\neq i\\ r[i]=t}} -w_{e<i} \end{pmatrix} \\
&= \sum_{t=j}^{r[e]-1} \begin{pmatrix} +p \sum\limits_{\substack{i\in N\\ e\neq i\\ r[i]=t+1}} (w_{i=e} - w_{i<e} - w_{e<i}) & + \sum\limits_{\substack{i\in N\\ e\neq i\\ r[i]=t+1}} w_{i<e} \\ -p \sum\limits_{\substack{i\in N\\ e\neq i\\ r[i]=t}} (w_{i=e} - w_{e<i} - w_{i<e}) & - \sum\limits_{\substack{i\in N\\ e\neq i\\ r[i]=t}} w_{e<i} \end{pmatrix}
\end{aligned}$$

Les sommes en facteur de p s'annulant entre elles pour toutes valeurs de $t \in [j+1; r[e-2]$, alles s'écrivent pour les valeurs j et $r[e]-1$ de la façon suivante :

$$\begin{aligned}
\delta_x(r,e,j) =& p \sum_{\substack{i\in N\\ e\neq i\\ r[i]=r[e]}} (w_{i=e} - w_{i<e} - w_{e<i}) - p \sum_{\substack{i\in N\\ e\neq i\\ r[i]=j}} (w_{i=e} - w_{e<i} - w_{i<e}) \\
&+ \sum_{t=j}^{r[e]-1} \left(\sum_{\substack{i\in N\\ e\neq i\\ r[i]=t+1}} w_{i<e} - \sum_{\substack{i\in N\\ e\neq i\\ r[i]=t}} w_{e<i} \right) \\
\delta_x(r,e,j) =& p \sum_{\substack{i\in N\\ e\neq i\\ r[i]=r[e]}} (w_{i=e} - w_{i<e} - w_{e<i}) - p \sum_{\substack{i\in N\\ e\neq i\\ r[i]=j}} (w_{i=e} - w_{e<i} - w_{i<e}) \\
&+ \sum_{\substack{i\in N\\ e\neq i\\ r[i]\in[j+1;r[e]]}} w_{i<e} - \sum_{\substack{i\in N\\ e\neq i\\ r[i]\in[j;r[e]-1]}} w_{e<i}
\end{aligned}$$

On fractionne les deux première sommes :

$$\begin{aligned}\delta_x(r,e,j) = \; & p \sum_{\substack{i\in N\\ e\neq i\\ r[i]=r[e]}} w_{i=e} && -p \sum_{\substack{i\in N\\ e\neq i\\ r[i]=r[e]}} (w_{i<e} + w_{e<i}) \\ & -p \sum_{\substack{i\in N\\ e\neq i\\ r[i]=j}} w_{i=e} && +p \sum_{\substack{i\in N\\ e\neq i\\ r[i]=j}} (w_{e<i} + w_{i<e}) \\ & + \sum_{\substack{i\in N\\ e\neq i\\ r[i]\in[j+1;r[e]]}} w_{i<e} && - \sum_{\substack{i\in N\\ e\neq i\\ r[i]\in[j;r[e]-1]}} w_{e<i}\end{aligned}$$

En identifiant terme à terme, on a $\delta_x(r,e,j) = \Delta_x^{inf}(r,e,j), \forall j \in [1; r[e]-1]$.

La démonstration permettant d'identifier $\delta_x(r,e,j)$ à $\Delta_x^{sup}(r,e,j)$ pour $j \in [r[e]+1; k]$ étant très similaire à celle effectuée ci-dessus entre $\delta_x(r,e,j)$ et $\Delta_x^{inf}(r,e,j)$ pour $j \in [1; r[e]]$, nous ne la détaillons pas ci-après. Nous nous contentons de dire que $\delta_x(r,e,j) = \Delta_x^{sup}(r,e,j), \forall j \in [r[e]+1; k]$.

On conclut alors que $\forall j \in [1;k]$, $\delta_x(r,e,j) = \Delta_x(r,e,j)$, et donc que $\delta_x(r,e,j)$ permet de décrire la variation de désaccords lors du déplacement de e dans un groupe existant à la position j. □

3.5.5 Pré-calcul des variations de distance en temps linéaire

Les fonctions $\delta_a(r,e,j)$ et $\delta_x(r,e,j)$ permettent de retourner les variations de distances. Nous donnons ici les clés pour calculer en temps linéaire (en nombre d'éléments) les valeurs de ces fonctions pour toutes les valeurs de j.

Considérons la fonction $\delta_a(r,e,j)$, le calcul de l'ensemble des valeurs se fait en deux phases, et avec une structure de données d qui est un tableau de $n+1$ réels. Dans un premier temps on calcule les valeurs de $\sigma_a^+(r,e,t)$, $\sigma_a^-(r,e,t)$, C^+ et C^- de façon que :

$$d[t] = \begin{cases} t < r[e] : & \sigma_a^+(r,e,t) \\ t = r[e] : & C_a^+(r,e) \\ t = r[e]+1 : & C_a^-(r,e) \\ t > r[e]+1 : & \sigma_a^-(r,e,t) \end{cases}$$

En détaillant :

$$d[t] = \begin{cases} t < r[e] : & \sum_{\substack{i \in N \\ e \neq i \\ r[i]=t}} w_{i<e} - w_{e<i} \\ t = r[e] : & \sum_{\substack{i \in N \\ e \neq i \\ r[i]=r[e]}} w_{i<e} + p \sum_{\substack{i \in N \\ e \neq i \\ r[i]=r[e]}} w_{i=e} - w_{e<i} - w_{i<e} \\ t = r[e]+1 : & \sum_{\substack{i \in N \\ e \neq i \\ r[i]=r[e]}} w_{e<i} + p \sum_{\substack{i \in N \\ e \neq i \\ r[i]=r[e]}} w_{i=e} - w_{e<i} - w_{i<e} \\ t > r[e]+1 : & \sum_{\substack{i \in N \\ e \neq i \\ r[i]=t-1}} w_{e<i} - w_{i<e} \end{cases}$$

Pour calculer ces valeurs en temps linéaire, il faut parcourir une unique fois l'ensemble des éléments, et pour chaque élément i, ajouter ses désaccords $w_{i<e}$, $w_{e<i}$ et $w_{e=i}$ dans les sommes adaptées en fonction de sa position $r[i]$. Considérons, par exemple, un élément i dont la position est $l = r[e] - 1$, les désaccords $w_{i<e}$ liés au fait de placer e avant i seront ajouté à $d[l]$, et les désaccords $w_{e<i}$ liés au fait de placer i avant e seront retirés de $d[l]$ (en effet placer l'élément e avant l'élément i retire les désaccords liés au fait que i était avant e).

La seconde phase consiste ensuite en le calcul des valeurs effectives de $\delta_a(r, e, j)$. Il suffit alors d'additionner les valeurs calculées précédemment de la façon suivante :

1: **Pour** j **de** $r[e] + 2$ **à** $nbGroupe(r) + 1$ **Faire**
2: $\quad d[j] \leftarrow d[j] + d[j-1]$
3: **Fin pour**
4: **Pour** j **de** $r[e] - 1$ **à** 1 **pas** -1 **Faire**
5: $\quad d[j] \leftarrow d[j] + d[j+1]$
6: **Fin pour**

On réalise le calcul de l'ensemble des valeurs de $\delta_x(r, e, j)$ en temps linéaire de façon semblable à $\delta_a(r, e, j)$.

On peut donc maintenant définir deux fonctions $\delta_a(r, e)$ et $\delta_x(r, e)$ calculant et retournant l'ensemble des variations de distance de la façon suivante :

Définition 3.5. Soit r un classement à égalité, e un élément, on définit la fonction $\delta_a(r, e)$ retournant sous forme d'un tableau de taille $nbGroupe(r) + 1$ la variation

de distance pour toute valeur de $j \in [1; nbGroupe(r) + 1]$:

$$\forall j \in [1; nbGroupe(r) + 1] : \delta_a(r, e)[j] = \delta_a(r, e, j)$$

Définition 3.6. Soit r un classement à égalité, e un élément, on définit la fonction $\delta_x(r, e)$ retournant sous forme d'un tableau de taille $nbGroupe(r)$ la variation de distance pour toute valeur de $j \in [1; nbGroupe(r)]$:

$$\forall j \in [1; nbGroupe(r)] : \delta_x(r, e)[j] = \delta_x(r, e, j)$$

3.5.6 Nouvelle implémentation et complexité

Nous présentons dans l'Algorithme 4 l'adaptation de l'Algorithme 3 en utilisant les fonctions $\delta_x(r, e)$ et $\delta_a(r, e)$ précédemment introduites. Son implémentation en Java est disponible en Appendice A.

L'évaluation de la différence de distance (ligne 10 en Algorithme 3 (original) se fait en temps linéaire, dans l'Algorithme 4 cette évaluation est effectué en temps constant. Par conséquence, la complexité de la recherche d'une position (lignes 9 à 15 dans les deux algorithmes) est donc réduite depuis $\mathcal{O}(n^2)$ vers $\mathcal{O}(n)$.

3.5.7 Conclusion

L'algorithme BioConsert est basé sur la modification gloutonne de plusieurs classements ($m + 1$, ou $m = |\mathcal{R}|$). La recherche d'une position dans r pour y déplacer un élément afin réduire la distance de Kendall-τ généralisée entre r et $\mathcal{R}$, avait une complexité en $\mathcal{O}(n^2)$ (ou $\mathcal{O}(n^3)$ pour une implémentation naïve) dans [CBDH11]. Nous avons présenté ici deux formules de récurrence permettant de pre-calculer l'ensemble les variations de distance lors du déplacement d'un élément en temps linéaire, et avons ainsi pu réduire la complexité de la recherche d'un position vers une complexité elle aussi linéaire. L'implémentation de cet algorithme prenant en compte cette amélioration est fournie en Appendice A.

Algorithme 4 BioConsert2015($\mathcal{R} \subseteq \mathcal{R}_n$, $p \in]0;1]$ paramètre du score de Kemeny)

$c \leftarrow [\{1,\ldots,n\}]$
Pour-Tout $r \in \mathcal{R} \cup \{\ [\{1,\ldots,n\}]\ \}$ **Faire**
  $encore \leftarrow 1$;
  Tant-que $encore > 0$ **Faire**
    $encore \leftarrow 0$; $change \leftarrow 0$
    Pour e **de** 1 **à** n **Faire** ..... // $\mathcal{O}(n^2)$
      $change \leftarrow 0$; $j \leftarrow 0$;
      $d = \delta_x(r, e)$ ..... // $\mathcal{O}(n)$
      Tant-que $j \leq nbGroup(r)$ **et** $change = 0$ **Faire** ..... // $\mathcal{O}(n)$
        Si $d[j] < 0$ **Alors** ..... // $\mathcal{O}(1)$
          $r \leftarrow changeDeGroupe(e, j)(r)$
          $change \leftarrow 1$
        FinSi
        $j \leftarrow j + 1$
      Fin Tant-Que
      $j \leftarrow 0$;
      $d = \delta_a(r, e)$ ..... // $\mathcal{O}(n)$
      Tant-que $j \leq nbGroup(r) + 1$ **et** $change = 0$ **Faire** ..... // $\mathcal{O}(n)$
        Si $d[j] < 0$ **Alors** ..... // $\mathcal{O}(1)$
          $r \leftarrow nouveauGroupe(e, j)(r)$
          $change \leftarrow 1$
        FinSi
        $j \leftarrow j + 1$
      Fin Tant-Que
      $encore = encore + change$
    Fin pour
  Fin Tant-Que
  Si $K^{(p)}(r, \mathcal{R}) < K^{(p)}(c, \mathcal{R})$ **Alors**
    $c \leftarrow r$
  FinSi
Fin pour
Retourner c.

3.6 Questions sur la complexité du problème

De nouvelles problématiques quant à l'interprétation des données ont mené à définir de nouvelle distances, puis par conséquent à adapter les algorithmes existants pour prendre en compte ces distances. Dans cette section, nous nous attachons à montrer que les problèmes formels découlant de l'utilisation de ces nouvelles distances sont au moins aussi difficiles que le problème de l'agrégation de classements sans égalité, justifiant ainsi la nécessité d'avoir recours à des heuristiques et des algorithmes approximations pour calculer des consensus.

Cette section s'organise de la façon suivante : dans un premier temps nous prouvons que le problème de l'agrégation de classements avec égalités est NP-difficile avec un nombre pair de classement supérieur à 3. Dans un second temps nous prouvons que le problème de l'agrégation de classements incomplets est NP-difficile.

3.6.1 Un problème au moins aussi difficile que sans les égalités

Le lemme et la preuve [BM15] ont été fait en collaboration avec Robin Milosz, étudiant de Sylvie Hamel (Professeur à l'Université de Montréal), qui a effectué un stage de recherche de deux mois (fevrier à avril 2015) au LRI. Ce lemme a permis dans le cadre de l'article [Bra+15] d'indiquer la complexité de l'agrégation de classements.

La complexité du problème dans le cas des égalités est un problème ouvert, cependant nous prouvons ici qu'en considérant la distance de Kendall-τ généralisée avec un paramètre $p = 1$, il est au moins aussi difficile que le problème d'agrégation de classements sans égalité. Les résultats présents et futurs de complexité du problème de l'agrégation de classements sans égalité sont donc valides pour l'agrégation de classements avec égalités.

Lemme 3.2. Le consensus optimal r^* d'un ensemble de classements $\mathcal{R} \subseteq \mathcal{R}_n$, en considérant la distance de Kendall-τ généralisée avec un paramètre $p = 1$, ne contient pas d'égalité si les classements de $\mathcal{R}$ n'en contiennent pas.

Preuve. Soit $\mathcal{R}$ un ensemble de m classements sur $[n]$ où aucun classement ne contient d'égalité. On note $c_{i<j}$ le nombre de classements dans $\mathcal{R}$ où i est avant j :

$$c_{i<j} = |\{r \in \mathcal{R} : r[i] < r[j]\}|$$

Soit r^* un consensus optimal de $\mathcal{R}$ au sens de la Kendall-τ généralisée avec $p = 1$:

$$\forall r \in \mathcal{R}_n, K^{(1)}(r^*, \mathcal{R}) \leq K^{(1)}(r, \mathcal{R})$$

Supposons par l'absurde que r^* contient au moins un groupe $G = \{1, \ldots, g\}$ contenant q éléments à égalité :

$$r^* = [\mathcal{B}_1, \ldots, \mathcal{B}_u, \underbrace{\{1, \ldots, g\}}_{G}, \mathcal{B}_v, \ldots, \mathcal{B}_k\}$$

Les éléments de G contribuent alors à la distance entre r^* et $\mathcal{R}$ pour :

$$K_G^{(1)}(r^*, \mathcal{R}) = \sum_{\substack{i,j \in G \\ i<j}} (c_{i<j} + c_{j<i})$$

Or $\mathcal{R}$ ne contient que des permutations, donc :

$$\forall i, j \in [n], i \neq j, c_{i<j} + c_{j<i} = m$$

donc

$$K_G^{(1)}(r^*, \mathcal{R}) = \sum_{\substack{i,j \in G \\ i<j}} m = \frac{q(q-1)}{2} m$$

Soit r' un classement identique à r^* dans lequel on a ensuite fragmenté G en groupes de 1 élément en lieu et place de G :

$$r' = [\mathcal{B}_1, \ldots, \mathcal{B}_u, \{1\}, \{2\}, \ldots, \{g\}, \mathcal{B}_v, \ldots, \mathcal{B}_k\}$$

Les éléments de G étant ordonnés dans r', pour chaque paire d'éléments $i, j \in G$ les désaccords comptabilisés dans la distance entre r' et $\mathcal{R}$ sont de façon exclusive soit $c_{i<j}$, soit $c_{j<i}$. On a donc

$$K_G^{(1)}(r', \mathcal{R}) \leq \sum_{i,j \in G, i<j} max(c_{i<j}, c_{j<i}) \leq \frac{q(q-1)}{2} m$$

Soit r'' un classement identique à r' dans lequel les éléments 1 et 2 sont ordonnés dans l'ordre majoritaire :

$$\text{si } c_{1<2} > c_{2<1} : r'' = [\mathcal{B}_1, \ldots, \mathcal{B}_u, \{1\}, \{2\}, \ldots, \{g\}, \mathcal{B}_v, \ldots, \mathcal{B}_k\}$$
$$\text{sinon } : r'' = [\mathcal{B}_1, \ldots, \mathcal{B}_u, \{2\}, \{1\}, \ldots, \{g\}, \mathcal{B}_v, \ldots, \mathcal{B}_k\}$$

Alors

$$\begin{aligned} K_G^{(1)}(r'', \mathcal{R}) &\leq min(c_{1<2}, c_{2<1}) + \sum_{\substack{1 \leq i<j \leq g \\ (i,j) \neq (1,2)}} max(c_{i<j}, c_{j<i}) \\ &\leq \frac{m}{2} + \left(\frac{q(q-1)}{2} - 1 \right) m \\ &< \frac{q(q-1)}{2} m \end{aligned}$$

Comme $K_G^{(1)}(r'', \mathcal{R}) < K_G^{(1)}(r^*, \mathcal{R})$, on a alors $K^{(1)}(r'', \mathcal{R}) < K^{(1)}(r^*, \mathcal{R})$ c'est-à-dire que r^* n'est pas une solution optimale, ce qui est en contradiction avec l'hypothèse que r^* contienne des égalités et soit optimal. On en conclut qu'en considérant la distance de Kendall-τ généralisée avec $p = 1$ et un ensemble de classements ne contenant pas d'égalité, il n'existe pas de consensus optimal contenant une ou des égalités. □

Corollaire 3.1. Le problème de l'agrégation de classements avec égalités en considérant un nombre pair d'au moins 4 classements est NP-difficile lorsque l'on considère la distance Kendall-τ généralisée avec $p = 1$.

Preuve. En se basant sur le Lemme 3.2, le problème de l'agrégation de permutations est un cas particulier du problème de l'agrégation de classements avec égalités lorsque l'on considère la distance de Kendall-τ généralisée. Comme le problème de l'agrégation de permutations est NP-difficile lorsque l'on considère un nombre pair ≥ 4 de permutations, il en est de même le problème de l'agrégation de classements avec égalités. □

3.6.2 Un problème NP-difficile pour des jeux de données incomplets

On cherche ici à savoir si le fait de considérer des jeux de données incomplets pour le problème de l'agrégation de classements avec la distance de Kendall-τ induite est NP-difficile dans le cas général.

Lemme 3.3. Le problème de l'agrégation de classements dans le cas où les permutations sont incomplètes,en utilisant la mesure de Kendall-τ induite est NP-difficile.

Preuve. On définit dans un premier temps une transformation polynomiale f de toute instance du problème du retrait minimal d'arcs dans un graphe orienté pour qu'il soit acyclique (*Minimum feedback arcset problem*) (FAS) en une instance du problème de l'agrégation de classements. Considérons un graphe orienté quelconque $G = (V, E)$, pour chaque arc $(u, v) \in E$ tel que $u \in V, v \in V$ on crée une permutation contenant deux éléments : $[u, v]$. Chaque arc est alors associé à

une unique permutation et réciproquement. L'ensemble des permutations créées forme alors un jeu de données incomplet I.

Prouvons maintenant que toute solution optimale pour l'instance G du FAS est une solution pour l'instance I du problème d'agrégation de permutations incomplètes.

Trouver un consensus optimal c au sens de la mesure de Kendall-τ induite pour l'instance I revient à trouver une permutation contenant l'ensemble des éléments de V. Par définition de la mesure de Kendall-τ induite, cette permutation minimise le nombre de désaccords entre c et les permutations de I. Cela revient donc à dire que c minimise le nombre de paires d'éléments i, j où i est placé avant j dans un classement de I alors que i est placé après j dans c. Sachant que chaque permutation de I ne contient que deux éléments, un désaccord correspond à une permutation. Le consensus optimal c minimise donc le nombre de permutations de I avec lesquelles il est en désaccord. Comme chaque permutation est associée à un unique arc du graphe G (par définition de f), le consensus optimal c minimise donc le nombre de d'arcs en désaccord avec lui-même. Nommons cet ensemble d'arcs F. Le consensus optimal c minimise donc la taille de l'ensemble F tel que $\forall (j, i) \in F$ i est classé avant l'élément j dans c. Les arcs de F sont alors les arcs à retirer à G afin de le rendre acyclique. □

3.7 Conclusion

Dans ce chapitre, nous sommes partis des besoins réels dans lesquels le problème de l'agrégation de classements permet d'obtenir des réponses, pour ensuite étudier les algorithmes permettant effectivement d'y répondre et finalement aborder différentes questions quant à la complexité du(des) problème(s) de l'agrégation de classements.

Plus précisément nous avons présenté un *guide à l'utilisateur* permettant de caractériser ses besoins et pouvant ainsi lui conseiller des mesures, distances et processus de normalisation à utiliser. Ces conseils permettent ainsi de calculer un consensus au plus près des besoins et des connaissances qu'un utilisateur a de ses données. Nous avons, dans ce cadre, formalisé deux nouvelles mesures, à savoir la *pseudo-distance de Kendall-τ* et la distance de *Kendall-τ généralisée induite*, afin de répondre respectivement à un besoin d'interprétation fine des égalités entre éléments et à un besoin de prise en compte des égalités dans les classements incomplets. Dans un second temps, nous nous sommes concentrés sur la problématique des égalités dans les classements et avons adapté certains algorithmes afin de considérer les égalités. Dans un troisième temps, et toujours dans le cadre des égalités entre éléments, nous avons introduit un nouvel algorithme pour calculer

des solutions optimales au problème de l'agrégation de classements en nous basant sur une traduction du problème en programmation en nombres entiers. Dans un quatrième temps, nous avons formalisé deux suites permettant, lors de leurs implémentations, de réduire la complexité en temps de l'algorithme BioConsert, et avons proposé une implémentation complète de ce dernier. Finalement, nous avons étudié la complexité du problème de l'agrégation de classements avec égalités et avons montré qu'il est au moins aussi difficile que celui sans égalité, à savoir NP-difficile pour un nombre pair de classement supérieur à 3. Nous avons aussi prouvé que le problème de l'agrégation de classements incomplets (avec égalités) était au moins aussi difficile que celui de retrait de cycles dans un graphe orienté, à savoir NP-difficile dans le cas général.

Le besoin d'algorithmes utilisant la mesure de Kendall-τ induite et la mesure de Kendall-τ généralisée induite n'a pas été traité dans ce chapitre. Les implémentations des algorithmes basés sur les distance de Kendall-τ classique ou généralisée s'adaptant aux classements incomplets. Il convient simplement d'ajouter un test vérifiant si un élément est effectivement présent dans un classement avant d'accéder à sa position. Cette adaptation a été faite sur l'algorithme BioConsert dans le cadre de [Sta+14], une application qui sera présentée au Chapitre 5.

Dans le chapitre suivant nous reprendrons l'ensemble des algorithmes introduits ici et les évaluerons de façon systématique sur des jeux de données réels et synthétiques.

Résumé du chapitre 3

Dans ce chapitre, nous avons proposé un *guide à l'utilisateur* qui, connaissant les besoins réels, permet à l'utilisateur de choisir les distances, mesures et approches de normalisation à utiliser pour pouvoir calculer un consensus. Ce guide a été illustré par plusieurs exemples et a mis en lumière de nouveaux besoins tels que la prise en compte d'égalité ou encore le fait que des jeux de données peuvent être incomplets.

Fort de l'identification des besoins et de la mise en avant de l'importance qu'il y a à prendre en compte les égalités entre éléments, nous avons proposé une méthodologie pour adapter les algorithmes à cette problématique et l'avons appliquée pour adapter une large majorité des algorithmes introduit au Chapitre 2.

Nous avons ensuite proposé ExactPNE, une traduction du problème de l'agrégation de classements avec égalités en programmation en nombres entiers, afin de pouvoir calculer des solutions optimales en prenant en compte le coût pour faire/rompre des égalités. Cet algorithme permettra donc de calculer des solutions exactes pour des instances du problème. Il permettra surtout de pouvoir comparer les résultats des approximations à la solution optimale en termes de désaccords via la distance de Kendall-τ, et ainsi de connaître le nombre de désaccords qu'une solution comporte par rapport à une solution optimale.

Considérant l'algorithme BioConsert, nous avons montré que la variation de distance lorsque l'on déplace un quelconque élément dans un classement peut se calculer en temps globalement linéaire pour tous les mouvements possibles. Cela nous a permis de réduire la complexité de l'algorithme d'un facteur linaire en n.

La prise en compte des égalités ou de l'absence d'éléments dans des classements (i.e : classements incomplets) est formellement différente du problème de l'agrégation de classements sans égalité tel que formulé par [Dwo+01]. Nous avons donc considéré la complexité du problème de l'agrégation de classements avec égalités et prouvé que ce dernier est au moins aussi difficile que lorsque l'on ne considère pas les égalités, à savoir que le problème est NP-difficile lorsqu'il y a un nombre pair de classements et qu'ils sont supérieurs à 3. Nous avons finalement montré que le problème de l'agrégation de classements lorsque les jeux de données sont incomplets est NP-difficile dans le cas général.

Chapitre 4

Évaluation de la performance et de la qualité des algorithmes d'agrégation de classements

Sommaire

Ce chapitre reprend des travaux que nous avons réalisés dans le cadre de l'article [Bra+15].

4.1 Introduction

Le chapitre précédent à fait état d'une problématique peu prise en compte par les algorithmes d'agrégation de classements : les *ex-æquo* entre éléments. Pour y répondre, nous avons proposé plusieurs adaptations d'algorithmes. L'objectif de ce nouveau chapitre est triple. Premièrement, il s'agit de mener une étude approfondie des algorithmes, tant en qualité des résultats qu'en rapidité à les produire sur des données réelles et synthétiques avec différentes caractéristiques. Deuxièmement, il s'agit de proposer des recommandations aux utilisateurs pour le choix d'un algorithme en fonction des caractéristiques des données. Troisièmement, il s'agit de proposer un outil permettant de reproduire et étendre les analyses faites dans ce chapitre à de nouveaux algorithmes.

Nous évaluons les algorithmes en fonction de plusieurs caractéristiques présentes dans les données réelles. La première d'entre elles est la taille d'un jeu de données (nombre d'éléments et de classements). Les évaluations sont conduites sur des données synthétiques dont la génération est faite uniformément afin d'éviter tout biais. Les résultats sont comparés à ceux obtenus sur des données réelles afin de mettre en lumière les questions qui ne peuvent être résolues en ne prenant en compte que la taille d'un jeu de données.

Nous avons vu au Chapitre 2 que plusieurs études font mention d'une qualité des résultats différente en fonction de la similarité intrinsèque des jeux de données. Nous évaluons dans ce chapitre les algorithmes avec des données générées spécifiquement pour avoir plusieurs degrés de similarités. Nous utilisons de plus une mesure de similarité standard permettant de réutiliser les conclusions liées au niveau de similarité. Tout comme avec les données synthétiques uniformes, nous comparons les résultats obtenus sur les données avec similarité aux données réelles afin de mettre en lumière une troisième caractéristique influant sur la qualité des résultats produits : le processus de normalisation, et *a fortiori* la taille des groupes où les éléments sont à égalité.

Nous étudions alors l'impact qu'ont deux processus de normalisation, à savoir la projection et l'unification. Nous montrons comment le premier peut être trop strict, et que le second peut revenir à mettre beaucoup d'éléments à égalité, altérant ainsi la qualité des résultats de certains algorithmes.

Ce chapitre s'organise de la façon suivante. Premièrement, nous présentons les algorithmes analysés, et les mesures et données (synthétiques et réelles) utilisées

dans les analyses. Nous présentons ensuite les analyses en les articulant autour de trois caractéristiques : la taille, la similarité et le processus de normalisation. Dans un troisième temps, nous proposons des recommandations dans le choix d'un algorithme. Finalement nous présentons un outil permettant de reproduire et étendre les analyses faites dans ce chapitre.

4.2 Environnement d'évaluation

Dans cette section, nous décrivons dans un premier temps les algorithmes évalués. Dans un deuxième temps, nous présentons les mesures utilisées pour évaluer la similarité des jeux de données, la qualité des résultats et le temps de calcul nécessaire pour produire un résultat. Nous poursuivons ensuite en présentant les jeux de données re-utilisés et nouvellement créés afin de comparer les algorithmes.

4.2.1 Algorithmes évalués

Les algorithmes que nous avons entièrement ré-implémentés et évalués sont indiqués en gras dans la Table 2.1 (p. 27). Certains algorithmes sont probabilistes et ne retournent donc pas nécessairement le même résultat à chaque exécution. Afin de les évaluer et de mettre en évidence la variabilité de la qualité des résultats de ces algorithmes, nous avons considéré 100 exécutions distinctes et choisi comme solution la meilleure calculée à travers ces diverses exécutions. Nous présentons les résultats avec le nom de l'algorithme avec le suffixe "Min" : *RepeatChoiceMin*, *KwikSortMin*.

4.2.2 Mesures

4.2.2.1 Mesurer la qualité d'un résultat

Pour évaluer la qualité des résultats, nous utilisons le *gap* [AM12 ; SZ09] (précédemment introduit en Section 2.4.1). Pour rappel, cette mesure permet d'avoir le pourcentage de désaccords en plus qu'a un consensus c par rapport au consensus optimal c^* :

$$gap = \frac{K(c\ , \mathcal{R})}{K(c^*, \mathcal{R})} - 1 \tag{4.1}$$

Dans le cas où il n'a pas été possible de calculer un consensus optimal, on calcule le $m - gap$. Dans ce cas, la distance d'un résultat produit par un algorithme est normalisée par la distance du meilleur consensus proposé par l'ensemble des algorithmes ayant effectivement pu calculer un consensus.

4.2.2.2 Mesurer le temps de calcul

Nous avons porté une attention toute particulière au protocole expérimental de mesure des temps de calculs afin d'assurer une comparaison des plus équitables. Les expériences ont été menées sur une machine avec quatre processeurs à double cœur Intel Xeon 3 GHz avec 16 Go de mémoire vive en utilisant Java 1.6.0_37, lpsolve 5.5.2.0, CPLEX 12.4 et Python 2.4.4.

Pour mesurer le temps d'exécution d'un algorithme, nous l'avons exécuté de nombreuses fois d'affilée de telle sorte que le temps nécessaire pour faire toutes les exécutions soit supérieur à deux secondes. Le temps d'exécution considéré pour un algorithme est alors le temps d'exécution global divisé par le nombre d'exécutions effectuées. Chaque mesure a été précédée par un temps de préchauffage où l'algorithme était relancé de nombreuses fois afin (a) d'observer le nombre d'exécutions nécessaires pour que la mesure du temps des exécutions successives dure plus de 2 secondes et (b) de s'assurer que toutes les classes java étaient déjà chargés dans la mémoire de la JVM.

Les implémentations n'utilisent qu'un *processus léger* (*thread*). Lpsolve (utilisé par Ailon$\frac{3}{2}$) et CPLEX (utilisé par ExactPNE) ont été utilisés dans leur configuration par défaut. Pour chaque algorithme, nous avons limité le temps de calcul à deux heures : une fois ce temps de calcul atteint, nous avons considéré que l'algorithme n'était pas en mesure de fournir une solution.

4.2.2.3 Mesurer la similarité d'un jeu de données

Certaines études ont observé que la similarité intrinsèque d'un jeu de données pouvait avoir une influence sur la qualité des consensus produits, ou sur le temps de calcul nécessaire pour produire un consensus. Cependant, la similarité fournie dans de tels travaux n'était qu'*a priori*, c'est-à-dire un paramètre de génération des jeux de données avec de la similarité, et n'était donc pas observable sur un quelconque jeu de données.

Afin de proposer des recommandations réutilisables, nous nous dotons d'une mesure pour évaluer cette similarité intrinsèque. Nous utilisons le taux de corrélation de Kendall (*Kendall–τ rank correlation coefficient*) [Ken38]. Cette mesure permet de quantifier la corrélation de deux permutations. L'extension de cette mesure afin de prendre en compte les égalités est définie ci-après.

Définition 4.1. Considérons $r_1, r_2 \in \mathcal{R}_n$ deux classements contenant n éléments. On rappelle que G est la distance de Kendall-τ généralisée (*cf.* Définition 1.13 p.17). Le taux de corrélation de Kendall généralisé est défini comme :

$$\tau(r_1, r_2) = \frac{\frac{1}{2}n(n-1) - 2G(r_1, r_2)}{\frac{1}{2}n(n-1)} \tag{4.2}$$

Afin de mesurer la similarité intrinsèque d'un jeu de données $\mathcal{R} = \{r_1, ...r_m\}$, nous calculons la corrélation moyenne de toutes les paires de classements du jeu de données.

Définition 4.2. La similarité intrinsèque d'un jeu de données $\mathcal{R} \subseteq \mathcal{R}_n$ contenant m classements est définie comme :

$$s(\mathcal{R}) = \frac{2}{m(m-1)} \sum_{i=1}^{m} \sum_{j=i+1}^{m} \tau(r_i, r_j) \tag{4.3}$$

4.2.3 Jeux de données réels

Nous avons d'abord considéré les jeux de données réels qui ont déjà été utilisés dans des travaux antérieurs (les quatre groupes de jeux de données sont en gras dans le Table 2.3 tout en étendant les expériences à un très grand ensemble d'algorithmes. Ces jeux de données sont de tailles différentes, à la fois en nombre d'éléments mais aussi en terme de nombre de classements par jeu de données. Ils ont été utilisés avec différentes normalisations. Finalement les études précédentes [AM12 ; BBN13 ; DK04] ont observé que la similarité d'un jeu de données pouvait avoir une influence sur la performance en qualité et en temps des algorithmes.

Afin de mieux comprendre l'influence et l'impact de ces trois caractéristiques, que sont la taille (nombre de classements, nombre d'éléments dans chaque classement), la similarité et la normalisation (unification ou projection), nous avons généré des jeux de données synthétiques. La génération de données synthétiques est décrite dans les deux sous-sections suivantes.

4.2.4 Jeux de données uniformément générés

Afin de se placer dans un cadre d'utilisation général, nous avons généré des jeux de données engendrés aléatoirement et uniformément : chaque classement d'un jeu de données à la même probabilité d'être engendré que tout les autres classements de même longueur. Nous avons utilisé pour cela le package MuPAD-Combinat [HT04] basé sur les travaux de [FZVC94].

Le nombre d'éléments a été choisi afin de contenir dans un jeu de données synthétique une quantité similaire d'éléments à celle des jeux de données réels. Le nombre de classements a été choisi pour imiter des ensembles de données du monde réel : la plupart d'entre eux contiennent moins de 10 classements. Plus précisément, nous avons produit des jeux de données de $m \in [3; 10]$ classements

avec des nombres n d'éléments différents : de 5 à 95 éléments de 5 en 5, puis de 100 à 500 éléments par étape de 100. Nous avons produit 100 jeux de données pour chaque paire $< m, n >$, ce qui donne un total de 16 800 jeux de données.

4.2.5 Jeux de données synthétiques avec des niveaux de similarité différents

Les résultats existants font apparaître le fait que la similarité intrinsèque d'un jeu de données aurait un impact sur le temps de calcul et/ou la qualité des résultats produits. Notre objectif est de mieux comprendre cet impact et que les conclusions faites soient liées à une similarité mesurable grâce à la mesure de similarité introduite en Section 4.2.2.3. Afin de faire le lien avec les résultats obtenus avec des données synthétiques uniformément, nous voulons que la génération de jeux de données puisse à la fois proposer des jeux de données similaires, et des jeux de données équivalents à ceux générés uniformément.

4.2.5.1 Présentation de la génération

Pour répondre aux contraintes relatives à la production de jeux de données dont la similarité peut varier depuis une similarité très forte jusqu'à une similarité équivalente à une génération de jeux de données uniformes, nous avons basé la génération sur une chaîne de Markov. Dans cette chaîne, chaque état correspond à un classement avec égalités et une transition entre deux états représente la modification d'un classements vers un autre. Ces modifications sont assurées par quatre opérateurs déplaçant un élément dans un groupe nouveau ou existant juste avant ou après sa position actuelle (*cf.* Définition 4.3). Ces opérateurs et leurs conditions d'utilisations assurent que la chaîne de Markov converge vers la distribution stationnaire uniforme. En considérant un classement initial r_s et un nombre de pas t, un jeu de données de m classements consiste à partir m fois de r_s dans la chaîne de Markov et d'ajouter l'état visité après t pas. Cette modélisation nous permet de générer des jeux de données, biaisés par l'état de départ r_s et le nombre de pas t. Ce nombre de pas fait dans la chaîne de Markov permet divers degrés de similarité vis-à-vis du classement d'origine, sachant que la distribution tend vers la distribution uniforme lorsque t tend vers l'infini.

Nous avons généré 1000 jeux de données contenant $m = 7$ classements sur $n = 35$ éléments avec un nombre de pas à parcourir dans la chaîne de Markov $t \in \{50, 100, 250, 500, 1000, 2500, 5000, 10000, 25000, 50000\}$. Le nombre de pas maximum a été estimé (*cf.* Section 4.2.5.4) comme étant suffisant pour avoir une génération uniforme à ϵ près, pour $\varepsilon = 0.01$.

Étudions maintenant les opérations de modification d'un classement, et montrons que ces opérations permettent que la chaîne de Markov générée avec ces derniers converge vers la distribution stationnaire uniforme.

4.2.5.2 Définition des opérations de base et de la chaîne de Markov

On définit ci-après les quatre opérations de base permettant de modifier un classement. On attache une importance particulière aux restrictions d'utilisation de ces opérations. Ces dernières permettent que la chaîne de Markov, telle que décrite en Définition 4.5, converge vers la distribution stationnaire uniforme.

Définition 4.3. On considère $r = [\mathcal{G}_1, \ldots, \mathcal{G}_k]$ un classement avec égalités. Soit $\mathcal{G}_i$ un groupe d'éléments de r contenant l'élément x. Nous définissons ci-après les quatre opérations de base ainsi que les restrictions conduisant à ne pas modifier r :

- **ajoutGauche(r,x)** : déplace l'élément x dans un nouveau groupe à gauche.
  Restriction : si $|\mathcal{G}_i| = 1$ alors r n'est pas modifié.
- **ajoutDroite(r,x)** : déplace l'élément x dans un nouveau groupe à droite.
  Restriction : si $|\mathcal{G}_i| = 1$ **ou** $|\mathcal{G}_\mathbf{i}| = \mathbf{2}$ alors r n'est pas modifié.
- **changeGauche(r,x)** : déplace l'élément x dans le groupe directement à gauche.
  Restriction : si $\mathcal{G}_i$ est le premier groupe alors r n'est pas modifié.
- **changeDroite(r,x)** : déplace l'élément x dans le groupe directement à droite.
  Restriction : si $\mathcal{G}_i$ est le dernier groupe **ou si** $\mathcal{G}_i$ **et** $\mathcal{G}_{i+1}$ **sont de taille 1** alors r n'est pas modifié.

Définition 4.4. On définit un mouvement comme étant une paire (x, f) où x est un élément et f une opération d'édition telle que définie précédemment : $f \in \{ajoutGauche, ajoutDroite, changeGauche, changeDroite\}$. Un mouvement est donc l'édition d'un quelconque classement r par l'opération f appliqué sur l'élément x.

On remarque qu'il y a 4 opérations et n éléments, il existe donc $4n$ mouvements, c'est-à-dire $4n$ applications différentes des opérations sur un classement donné.

Définition 4.5. En considérant $\mathcal{R}_n$ l'ensemble des classements avec égalités contenant n éléments, on construit une chaîne de Markov où :
- chaque état correspond à un unique classement, et chaque classement correspond à un unique état
- il existe une transition d'un classement c_1 vers un classement c_2 *si et seulement si* il existe au moins un *mouvement* applicable sur un élément de c_1 pour former c_2
- la probabilité d'utilisation d'un *mouvement* en considérant un classement est $\frac{1}{4n}$
- la probabilité de transition $p(e,e) = 1 - \sum_{e' \neq e} p(e,e')$

Les opérations de modification de classement et la chaîne de Markov étant maintenant définies, nous prouvons que cette chaîne converge vers la distribution stationnaire uniforme dans la sous-section suivante.

4.2.5.3 Preuve de convergence vers la distribution stationnaire uniforme

Pour prouver la convergence vers la distribution uniforme, nous prouvons dans un premier temps qu'elle tend vers une unique distribution, c'est-à-dire qu'elle est irréductible (*cf.* Lemme 4.1), et apériodique (*cf.* Lemme 4.2). Dans un deuxième temps, nous prouvons que chaque transition a une transition inverse (*cf.* Lemme 4.3) et que chaque transition et son inverse ont la même probabilité de transition (*cf.* Lemme 4.4). La chaîne de Markov converge donc vers la distribution uniforme.

Lemme 4.1. La chaîne de Markov (*cf.* Définition 4.5) est irréductible.

Preuve. À partir de chaque classement, on peut aboutir au classement où tous les éléments sont dans le même groupe en utilisant changeGauche autant que nécessaire. En partant de ce classement, on peut arriver à tous les classements en utilisant ajoutGauche et changeGauche. Chaque état étant connecté, la chaîne est irréductible. □

Lemme 4.2. La chaîne de Markov (*cf.* Définition 4.5) a pour chaque état une transition pointant sur lui-même.

Preuve. Par définition de l'opération changeGauche, son application sur un élément en première position dans un classement ne change pas le classement. Cela assure que chaque état/classement ait une transition vers lui-même associé à une probabilité non-nulle. □

Nous avons ainsi prouvé que la chaîne de Markov définie en 4.5 est ergodique. Prouvons maintenant que la distribution vers laquelle elle converge est la distribution uniforme.

Lemme 4.3. La chaîne de Markov (*cf.* Définition 4.5) contient pour chaque transition une transition inverse.

Preuve. On considère un classement $r = [\mathcal{G}_1, \ldots, \mathcal{G}_k]$, et un élément $x \in \mathcal{G}_i$. On applique une opération sur x. S'il y a une altération de r produisant r' (avec $x \in \mathcal{G'}_j$), alors il existe toujours une opération inverse pour revenir de r' à r. Ces opérations inverses sont énumérées ci-après.

Opération	Opération inverse	Condition
ajoutDroite(x)	changeGauche(x)	toujours
changeDroite(x)	changeGauche(x)	toujours
changeGauche(x)	changeDroite(x)	si $\lvert\mathcal{G}_i\rvert > 1$
changeGauche(x)	ajoutDroite(x)	si $\lvert\mathcal{G}_i\rvert = 1$ et $\lvert\mathcal{G'}_j\rvert > 2$
changeGauche(x)	ajoutGauche(y)	si $\lvert\mathcal{G}_i\rvert = 1$ et $\lvert\mathcal{G'}_j\rvert = 2$ alors $\mathcal{G'}_j = \{x, y\}$, et on applique l'opération inverse sur y
ajoutGauche(x)	changeDroite(x)	si $\lvert\mathcal{G}_i\rvert > 2$
ajoutGauche(x)	changeGauche(z)	si $\lvert\mathcal{G}_i\rvert = 2$ alors $\mathcal{G}_i = \{x, z\}$, et on applique l'opération inverse sur z

□

Lemme 4.4. Soit deux classements distinct c_1 et c_2. Il existe au plus un mouvement (*cf.* Définition 4.4) permettant de transformer c_1 en c_2, et donc la probabilité de transition $p(c_1, c_2) = \frac{1}{4n}$.

Preuve. On considère un classement $r = [\mathcal{G}_1, \ldots, \mathcal{G}_k]$, et un élément $x \in \mathcal{G}_i$. On applique une opération sur x, pour produire un classement r' différent de r.

Considérons l'opération ajoutGauche(r, x)

- si $|\mathcal{G}_i| = 1$, alors cette fonction est sans effet.
- si $|\mathcal{G}_i| = 2$, alors on considère que $\mathcal{G}_i = \{x, y\}$. L'opération ajoutGauche a pour effet de mettre x et y dans deux groupes singletons différents. Pour aboutir au même résultat , il faut appliquer ajoutDroite(y) or par définition cette fonction est sans effet si l'élément est dans un groupe de cardinalité 2.
- si $|\mathcal{G}_i| > 2$, placer x dans un groupe singleton à gauche de $\mathcal{G}_i$ est uniquement faisable par l'opération ajoutGauche(x), il n'existe en effet pas d'opération permettant de déplacer plusieurs éléments en une unique utilisation.

Il n'y a donc aucune opération permettant de reproduire la modification de $ajoutGauche(x)$.

Considérons l'opération ajoutDroite(r, x)

- si $|\mathcal{G}_i| = 1$, alors cette fonction est sans effet.
- si $|\mathcal{G}_i| = 2$, alors cette fonction est sans effet, par définition de l'opérateur (*cf.* Définition 4.3)
- si $|\mathcal{G}_i| > 2$, placer x dans un groupe singleton à droite de $\mathcal{G}_i$ est uniquement faisable par l'opération ajoutDroite(x), il n'existe en effet pas d'opération permettant de déplacer plusieurs éléments en une unique utilisation.

Il n'y a donc aucune opération permettant de reproduire la modification de $ajoutDroite(x)$.

Considérons l'opération changeGauche(r, x)

- si $|\mathcal{G}_i| = 1$, alors le groupe a été fusionné avec le groupe directement à gauche, le groupe $\mathcal{G}_{i-1}$. Pour reproduire cette modification il faut que $\mathcal{G}_{i-1}$ soit fusionné avec $\mathcal{G}_i$:
  - si $|\mathcal{G}_{i-1}| > 1$ il faut déplacer plusieurs éléments, ce qui n'est pas possible avec les opérateurs.
  - si $|\mathcal{G}_{i-1}| = 1$ il faut appliquer *changeDroite* sur l'élément de $\mathcal{G}_{i-1}$, or par définition de l'opération *changeDroite*, cette fonction reste sans effet sur un groupe singleton dont le groupe mitoyen à droite est lui aussi un singleton (*cf.* Définition 4.3).
- si $|\mathcal{G}_i| > 1$, il faut déplacer plus d'un élément ce qui n'est pas possible avec les opérateurs.

Il n'y a donc aucune opération permettant de reproduire la modification de $changeGauche(x)$.

Considérons l'opération changeDroite(r, x)

- si $|\mathcal{G}_i| = 1$, alors cette fonction est sans effet.
- si $|\mathcal{G}_i| > 1$, il faudrait déplacer plus d'un élément ce qui n'est pas possible avec les opérateurs.

Il n'y a donc aucune opération permettant de reproduire la modification de $changeDroite(x)$.

On conclut donc que pour toute paire de classements distincts c_1, c_2, il existe au plus un mouvement permettant d'éditer c_1 en c_2. Par définition de la chaîne de Markov, la probabilité d'utilisation d'un *mouvement* est de $\frac{1}{4n}$, on en déduit donc que toute transition entre deux états distincts est elle aussi de $\frac{1}{4n}$. □

On remarque que la transition permettant de rester dans l'état courant a une probabilité supérieur ou égale à $\frac{2}{4n}$. En effet changeGauche sur le(s) premier(s) élément(s) et changeDroit sur le(s) dernier(s) élément(s) sont par définition sans effet, il y a donc une probabilité de au moins $2 \times \frac{1}{4n}$ de rester dans l'état courant.

Théorème 4.1. La chaîne de Markov définie en 4.5 converge vers la distribution stationnaire uniforme.

Preuve. D'après les Lemmes 4.1 et 4.2 la chaîne de Markov converge vers une unique distribution. D'après le Lemme 4.3, chaque transition à une transition inverse. D'après le Lemme 4.4, chaque transition entre deux états distincts est réalisable dans le cadre d'au plus un mouvement, et a donc une probabilité de transition constante de $\frac{1}{4n}$.

On peut donc conclure que la chaîne de Markov, telle que définie en 4.5, converge vers la distribution stationnaire uniforme. □

On remarque que si la définition de la chaîne de Markov avait inclus que chaque transition ait une probabilité de transition de $\frac{1}{4n}$, le Lemme 4.4 aurait été superflu. Par conséquence, le fait qu'une transition est associée à un seul mouvement serait alors resté inconnu. Lors de la mise en œuvre de cette chaîne de Markov il aurait alors fallu, lorsque l'on visite un état, énumérer l'ensemble des transitions au départ de cet état puis d'emprunter une transition en respectant la probabilité de $\frac{1}{4n}$ si la transition est sortante. Connaître le fait qu'une transition est réalisable dans le cadre d'un unique mouvement permet de s'affranchir de cette coûteuse énumération en tirant uniformément un élément et une opération d'édition.

4.2.5.4 Temps pour atteindre la distribution stationnaire uniforme

Nous cherchons ici à estimer expérimentalement le nombre de pas à faire dans la chaîne de Markov afin d'avoir une distribution uniforme à ε près. Suivant [Den02], une chaîne de Markov (E, V) a atteint une distribution uniforme à ε près lorsque la différence relative entre chaque probabilité et la probabilité uniforme $p_u = \frac{1}{|E|}$ est inférieure à ε :

$$\max_{e \in E} \frac{|p(e) - p_u|}{p_u} \leq \varepsilon$$

Nous avons dans un premier temps construit les matrices de transition de la chaîne de Markov pour plusieurs tailles de classements avec n éléments (*cf.* Table 4.1).

Nombre d'éléments (n)	2	3	4	5	6
Nombre d'états	3	13	75	541	4683

TABLE 4.1 – Nombre d'états dans la chaîne de Markov en fonction du nombre de d'éléments.

En partant d'un état initial aléatoirement choisi, on calcule le nombre de pas à faire avant de s'approcher de la distribution uniforme à $\varepsilon \in \{0.02, 0.01, 0.005, 0.002, 0.001\}$ près. La convergence observée, on note le nombre d'éléments, l'ε attendu, l'ε_m mesuré (donc $\varepsilon_m \leq \varepsilon$), et finalement le nombre de pas parcourus. L'opération a été répétée 200 fois pour chaque paire $< n, \varepsilon >$, avec des états initiaux tirés aléatoirement (avec remise) en utilisant le logiciel Octave [Oct14].

La forme de la fonction retournant le nombre de pas en fonction de n et ε a été déterminé expérimentalement en utilisant SageMath [Ste+15]. Nous avons comparé plusieurs modèles énumérés en Figure 4.2 et avons observé que (1) le nombre de pas est plus susceptible d'évoluer polynomialement en fonction de n, et que (2) le nombre de pas est plus susceptible d'évoluer de façon logarithmique en fonction de ε.

On présente en Figure 4.3 la fonction choisie et les valeurs ajustées. On présente en Figure 4.1 les valeurs du nombre de pas mesurées pour différents n et ε ainsi que le tracé de la fonction ajustée pour différents n et ε. En Table 4.2 différents nombres de pas sont estimés pour atteindre l'uniformité à ε-près en fonction de n.

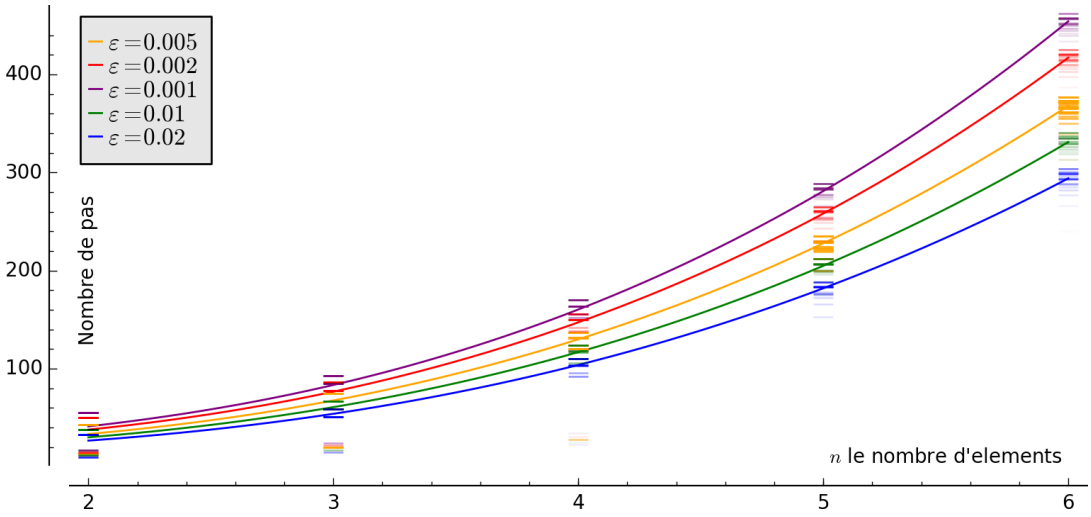

FIGURE 4.1 – Nombre de pas à parcourir pour s'approcher de la distribution uniforme à ε-près et fonction ajustée. Chaque observation du nombre de pas pour un ε donné est représentée par un tiret horizontal. À noter que chaque tiret est dessiné avec une transparence de 99%, les points très colorés sont donc un grand empilement de points, les points faiblement colorés étant quelque points superposés.

fonction	erreur moyenne
$pas(n,\varepsilon) = (a - b * log(\varepsilon)) * (n^p + s)$	7.82
$pas(n,\varepsilon) = (a - b * log(\varepsilon)) * (e^n + s)$	8.11
$pas(n,\varepsilon) = (a - b * \varepsilon) * (n^p + s)$	13.25
$pas(n,\varepsilon) = (a - b * \varepsilon) * (e^n + s)$	13.37

FIGURE 4.2 – Fonctions étudiées et erreur moyenne en nombre de pas par rapport à chacune des données utilisées pour ajuster la courbe.

$$pas(n,\varepsilon) = (a - b * log(\varepsilon)) * (n^p + s)$$

variable	valeur
a	0.5190601470597632
b	0.3242877158672346
s	7.230990807278927
p	2.8229822610996025

FIGURE 4.3 – Fonction choisie pour estimer le nombre de pas à parcourir dans la chaîne, et valeurs des paramètres ajustées avec SageMath.

# éléments	# pas
2	28
3	59
4	115
5	203
6	331
35	45998
100	890633

TABLE 4.2 – Nombre de pas requis selon la fonction ajustée pour $\varepsilon = 0.01$ et différentes valeurs de n.

Étape	Exemple	Jeux de données génères
Jeu de données original	$\left\{ \begin{array}{l} [\{\mathbf{A}\},\{\mathbf{B},\mathbf{C}\},\{F\},\{D\},\{E\}] \\ [\{\mathbf{D}\},\{\mathbf{A},\mathbf{E}\},\{F\},\{B\},\{C\}] \\ [\{\mathbf{A}\},\{\mathbf{C}\},\{D\},\{B\},\{E,F\}] \end{array} \right\}$	7 classements 100 éléments
↓ Conservation des $k = 2$ premiers éléments ↓	$\left\{ \begin{array}{l} [\{A\},\{B,C\}] \\ [\{D\},\{A,E\}] \\ [\{A\},\{C\}] \end{array} \right\}$	$k \in [1;35]$
Application du processus d'unification	$\left\{ \begin{array}{l} [\{A\},\{B,C\},\{\mathbf{D},\mathbf{E}\}] \\ [\{D\},\{A,E\}\{\mathbf{B},\mathbf{C}\}] \\ [\{A\},\{C\},\{\mathbf{B},\mathbf{D},\mathbf{E}\}] \end{array} \right\}$	Jeu de données avec $n = 35$ éléments

FIGURE 4.4 – Étapes suivies pour la génération de données synthétiques unifiés. Dans cet exemple, un jeu de données avec 3 classements et 6 éléments est généré, ensuite les $k = 2$ premiers éléments sont conservés. Finalement, le processus d'unification est appliqué et permet d'obtenir un jeu de données sur les mêmes 5 éléments.

4.2.6 Jeux de données synthétiques unifiés avec des similarités

Nous étudions maintenant l'impact qu'a le processus d'unification (introduit en (*cf.* Section 2.3.2 p.38)) sur la qualité des consensus calculés par les différents algorithmes lorsqu'il est utilisé sur des jeux de données qui ne sont pas sur les mêmes éléments (BioMedical [CBDH11]), ou encore lorsque l'ensemble de données est uniquement constitué de k premiers éléments de chaque classement (WebSearch [Dwo+01 ; SZ09]).

Nous avons simulé ce deuxième cas d'utilisation et avons généré des jeux de données avec différents niveaux de similarité tel qu'il a été fait dans la section précédente, en conservant uniquement les k premiers éléments puis en appliquant le processus d'unification. La génération de ces données est illustrée en Figure 4.4.

Nous avons généré 1000 jeux de données avec $m = 7$ classements contenant $n = 100$ éléments généré avec la modélisation par chaîne de Markov introduite précédemment en utilisant un nombre de pas à parcourir $t \in \{1\,000, 2\,500, 5\,000, 10\,000, 25\,000, 50\,000, 100\,000, 250\,000, 500\,000, 1\,000\,000\}$. Soit $k \in [1;35]$, les premiers éléments de chaque classement sont conservés de façon à obtenir, après

unification, des jeux de données contenant $n = 35$ éléments.

4.3 Évaluations des algorithmes

Dans cette section, nous présentons les résultats obtenus par différents algorithmes d'agrégation de classements en utilisant différents types de jeux de données. Dans un premier temps, nous évaluons la qualité des résultats produits par les algorithmes et leur consommation en temps en fonction de la taille des jeux de données. Pour ce faire, nous utilisons des jeux de données générés de manière uniforme. Dans un deuxième temps, nous prenons en compte différents niveaux de similarité avec l'aide des jeux de données synthétiques *avec similarité*. Dans un troisième temps, nous analysons l'impact des processus de normalisation (l'unification et la projection) sur les jeux de données données réelles. Finalement, nous analysons l'impact du processus d'unification sur la qualité des résultats produits.

Dans chaque analyse, nous étudions également l'impact des résultats obtenus sur les jeux de données réels avec les mêmes connaissances (la taille, la similarité, la normalisation), ce qui nous permet de mettre en évidence les informations nécessaires pour bien comprendre le comportement des algorithmes dans des contextes réels.

4.3.1 Évaluation en fonction de la taille des jeux de données

4.3.1.1 Évaluation qualitative

Nous avons mené une première série d'expériences sur des jeux de données générés de manière uniforme. Nous considérons des jeux de données avec $m \in [3; 10]$ classements et $n \leq 60$ éléments (60 étant le plus grand nombre d'éléments pour lesquels le consensus optimal peut être calculé dans un délai raisonnable). Les résultats sont présentés en Table 4.4. Quatre points sont à souligner.

Tout d'abord, à la fois pour les jeux de données synthétiques et mais aussi réels (*cf.* Table 4.3 et Table 4.4), BioConsert fournit les meilleurs résultats dans la très grande majorité des cas (88.56%) et dans 33.94% des cas il surpasse strictement les autres algorithmes. Dans 68.01% des cas, les solutions produites sont optimales. Sur les jeux de données réels, BioConsert fournit les meilleurs résultats pour 91.8% des jeux de données.

Deuxièmement, Ailon$\frac{3}{2}$ est une approximation très efficace car elle a un *gap* proche de 0, cependant l'implémentation actuelle de l'algorithme ne passe pas à l'échelle : pour $n > 45$ aucun résultat n'est disponible. Sur les jeux de données synthétiques, Ailon$\frac{3}{2}$ et BioConsert fournissent des résultats avec une qualité

TABLE 4.3 – Pour chaque algorithme nous présentons le **gap** moyen obtenu sur chaque type de jeux de données réelles, ainsi que son rang. Pour les jeux de données WebSearch unifié nous présentons le $\mathbf{m-gap}$ car la taille des données ne permet pas d'obtenir les solutions exactes nécessaires au calcul du *gap*..

	WebSearch		F1		SkiCross		BioMed.	
Algorithm	(%) Proj	Unif ($m-gap$, %)	(%)Proj	(%)Unif	(%) Proj	(%)Unif	(%)Unif	$\%1^{st}$
Ailon$\frac{3}{2}$	0(# 1)	—	0(# 1)	16(# 4)	—	—	16,8(# 6)	17,1%
BioConsert	0(# 1)	.00026(# 1)	.0015(# 2)	0(# 1)	0,11(# 1)	0(# 1)	0,17(# 2)	91,8%
BordaCount	10,9(# 8)	55,9(# 8)	3,7(# 6)	30,2(# 8)	4(# 5)	27,7(#10)	20,7(# 9)	0,4%
Copeland-Method	10,9(# 8)	55,6(# 7)	3,7(# 6)	30,2(# 9)	4(# 5)	26,7(# 8)	20,3(# 8)	0,4%
FaginLarge	10,9(# 7)	55,9(# 9)	15,1(# 9)	17,8(# 6)	3,8(# 4)	23,9(# 7)	31,7(#11)	
FaginSmall	4,6(# 5)	57(#11)	3,7(# 5)	31,6(#10)	3,2(# 3)	27,2(# 9)	23,4(#10)	2,5%
KwikSort	5,4(# 6)	33,5(# 3)	1,4(# 4)	3,2(# 3)	4,5(# 7)	16,7(# 5)	2,4(# 3)	9,0%
KwikSort-Min	0,2(# 3)	32,2(# 2)	0(# 3)	0,2(# 2)	1,8(# 2)	15,1(# 3)	0,16(# 1)	61,0%
MEDRank (h=0.5)	12,5(#11)	45,2(# 6)	17,5(#10)	17,2(# 5)	6,5(# 8)	13,7(# 2)	7,5(# 4)	4,9%
MEDRank (h=0.7)	12,4(#10)	37,3(# 4)	24,8(#11)	20,7(# 7)	9,6(# 9)	15,7(# 4)	18,3(# 7)	0,2%
Pick-a-Perm	44,4(#13)	41,4(# 5)	27(#12)	39(#11)	19,1(#10)	18,8(# 6)	68,1(#13)	
Repeat-Choice	24,8(#12)	57,5(#12)	27,5(#13)	54,5(#13)	23,3(#12)	34,6(#12)	31,9(#12)	
Repeat-ChoiceMin	1,2(# 4)	56,4(#10)	8,9(# 8)	40(#12)	19,1(#10)	30,2(#11)	16,4(# 5)	4,7%
# datasets	36	37	48	48	1	1	319	490

Algorithme	rang	*gap* moyen	%*gap* =0	%premier
Ailon$\frac{3}{2}$	# 2	0,38%	63,15%	64.31%
BioConsert	# 1	0,03%	68,01%	88.56%
BordaCount	# 7	5,6%	2,53%	2.17%
CopelandMethod	# 5	4,4%	3,69%	3.69%
FaginSmall	# 6	4,7%	3,21%	3.21%
FaginLarge	# 9	10,8%	0,44%	0.44%
KwikSortMin	# 3	1,2%	23,98%	24.02%
KwikSort	# 4	4,1%	4,14%	4.13%
MEDRank(0.5)	#10	12,9%	0,62%	0.62%
MEDRank(0.7)	#11	17,2%	0,41%	0.41%
Pick-a-Perm	#13	20%	0,84%	0.84%
RepeatChoice	#12	17,6%	0,02%	0.02%
RepeatChoiceMin	# 8	9,7%	5,8%	5.84%

Table 4.4 – Pour chaque algorithme, rang en terme de *gap* moyen, pourcentage des jeux de données où l'algorithme a trouvé un consensus optimal, pourcentage des jeux de données où l'algorithme est le meilleur (*ex-æquo compris*). Les jeux de données considérés sont ceux générés uniformément sur $n \leq 60$ éléments et $m \in [3; 10]$.

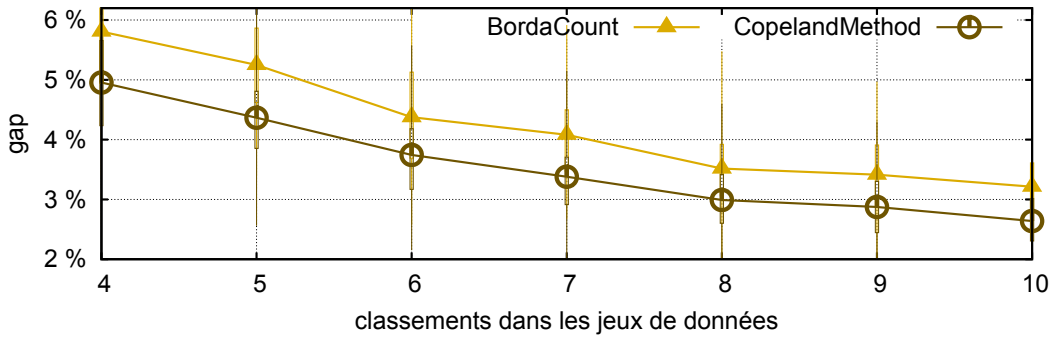

Figure 4.5 – Valeurs de *gap* pour BordaCount et CopelandMethod en utilisant les jeux de données synthétiques avec n=35 éléments pour différentes valeurs de m.

similaire : Ailon$\frac{3}{2}$ est meilleur que BioConsert dans 14.39% des jeux de données, alors qu'il est moins bon que BioConsert dans $18,67\%$ d'entre eux.

Troisièmement, les algorithmes positionnels BordaCount et CopelandMethod ont un comportement intéressant sur les jeux de données synthétiques. Lorsqu'ils sont comparés aux autres algorithmes, les nombres de désaccords moyens de

leurs solutions décroissent lorsque le nombre d'éléments augmente. En considérant les jeux de données avec $m = 7$ classements, sur les jeux de données de 20 éléments CopelandMethod (respectivement BordaCount) est classé $8^{ième}$ (respectivement $10^{ième}$) alors qu'il est $3^{ième}$ (respectivement $4^{ième}$) avec des jeux de données de 500 éléments.

Quatrièmement, l'amélioration de BordaCount et CopelandMethod est observée en Figure 4.5 lorsque l'on augmente le nombre de classements.

Finalement, nous avons évalué le comportement de MEDRank en variant la valeur de son seuil, ce dernier étant le pourcentage de classements dans lesquels il faut avoir vu un élément avant de l'ajouter au consensus calculé. En observant le *gap* pour différentes valeurs du seuil, nous avons remarqué que MEDRank y est très sensible. Plus précisément, les valeurs de seuil plus élevées que celle par défaut (seuil de 0.5) ne conduisent pas à une amélioration de la qualité du consensus prévu, ni en temps réel, ni dans jeux de données synthétiques. Dans 76.37% des jeux de données synthétiques un seuil de 0.5 fournit les meilleurs résultats. C'est donc la valeur de seuil à préférer lors de l'utilisation de MEDRank.

Alors que certains algorithmes montrent un comportement identique entre les jeux de données synthétiques et réels (*cf.* Table 4.3 et Table 4.4), d'autres ont des comportements différents. Trois observations soulignent le besoin d'informations plus précises sur les jeux de données afin de comprendre les situations où les algorithmes produisent des consensus de qualité.

Tout d'abord, lors de l'examen des variantes FaginLarge et FaginSmall, ce dernier retourne de meilleurs résultats que FaginLarge dans 99.59% des jeux de données synthétiques. Cette observation n'est pas répétée sur les jeux de données réels, en effet les deux versions retournent des résultats en moyenne similaires : FaginSmall est meilleurs que FaginLarge dans seulement 49.52% des jeux de données.

Deuxièmement, BordaCount et CopelandMethod semblent s'améliorer lorsque le nombre d'éléments et de classements augmente dans les jeux de données synthétiques. Ce comportement ne peut être expliqué uniquement par la taille des jeux de données.

Troisièmement BordaCount et CopelandMethod, produisent sur les jeux de données synthétiques des résultats avec un gap proche de 5%, cependant cette tendance n'est pas observée dans les jeux de données réels.

Prendre en compte une autre caractéristique des jeux de données, à savoir la similarité, pourrait aider à comprendre les comportements des algorithmes et sera fait en Section 4.3.2.

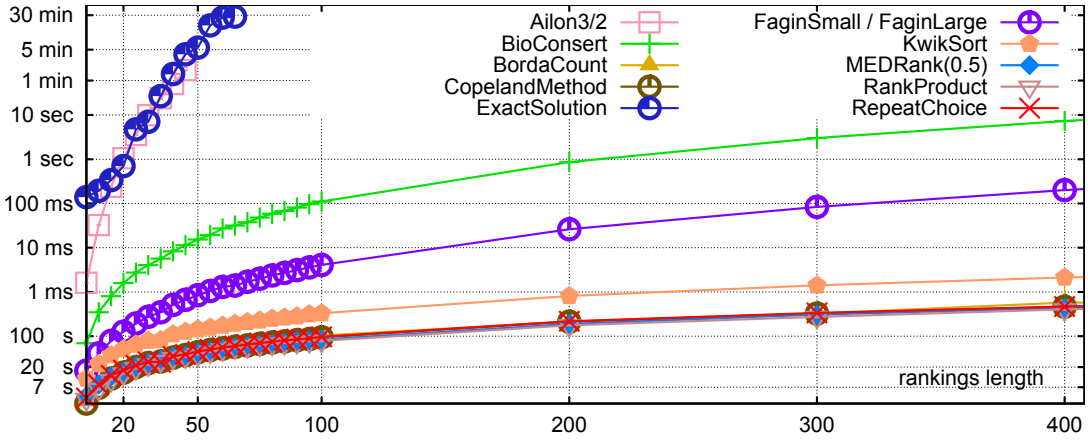

FIGURE 4.6 – Temps de calcul pour un nombre d'éléments $\mathbf{n} \in [\mathbf{5}; \mathbf{400}]$, et pour 7 classements. On note que BORDACOUNT, COPELANDMETHOD, MEDRANK et REPEATCHOICE ne peuvent pas être distingués les uns des autres. Un agrandissement de la partie inférieure gauche du graphique est fournie en Figure 4.7.

4.3.1.2 Temps de calcul expérimental sur des jeux de données générés de manière uniforme

Nous considérons maintenant le temps de calcul pour différentes valeurs de n le nombre d'éléments dans les jeux de données et un nombre de classements $m = 7$ fixe.

Tout d'abord, on remarque en Figure 4.7 que le temps de calcul moyen nécessaire aux algorithmes positionnels MEDRank, COPELANDMETHOD, REPEATCHOICE et BORDACOUNT pour retourner une solution est très faible : pour les jeux de données de $n = 400$ ils prennent en moyenne respectivement $436\mu s$, $463\mu s$, $468\mu s$ et $574\mu s$. Avec les données synthétiques, nous avons observé que lorsque n grandit, MEDRANK, COPELANDMETHOD, BORDACOUNT retournent (encore) des consensus de bonne qualité par rapport aux autres algorithmes tout en étant bien plus rapides. Ils sont donc de très bons candidats pour les très grands jeux de données.

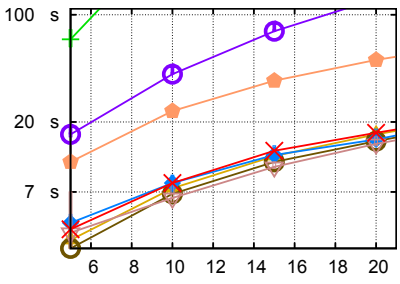

FIGURE 4.7 – Agrandissement de la Figure 4.6

Deuxièmement, en ne considérant que les deux algorithmes retournant des résultats de grandes qualité, BIOCONSERT surpasse strictement AILON$\frac{3}{2}$, par exemple pour $n = 40$ éléments ils prennent respectivement $0.0083s$ et $50.373s$.

De l'analyse conjointe de la Table 4.3, de la Table 4.4 et de la Figure 4.6, il

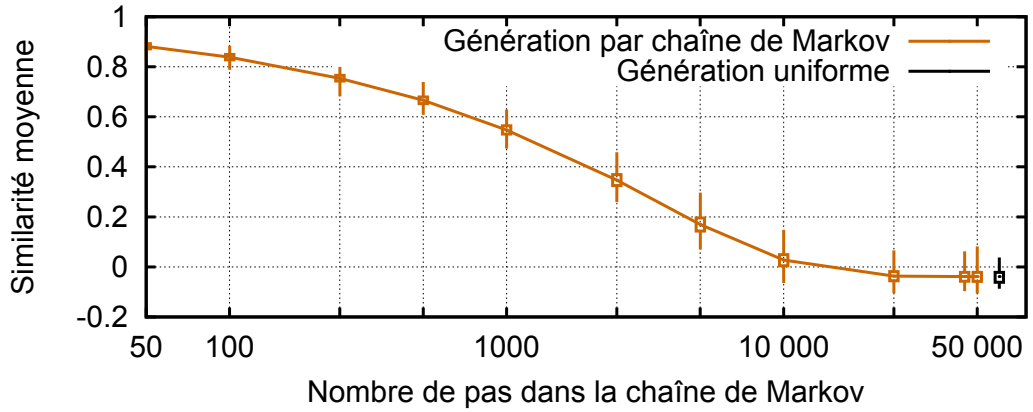

FIGURE 4.8 – Similarité moyenne des jeux de données générés par chaîne de Markov en fonction du nombre de pas utilisés lors de la génération.

apparaît que BIOCONSERT est capable de fournir des résultats de qualité en un temps de calcul très raisonnable alors que les algorithmes positionnels, comme BORDACOUNT ou COPELANDMETHOD, peuvent fournir des réponses très rapidement mais avec une qualité inférieure.

4.3.2 Évaluation en fonction de la similarité des jeux de données

Sur les 10^5 jeux de données produits et utilisés dans cette expérience, un consensus optimal a été trouvé dans 99.91% des jeux de données par EXACTPNE avec le temps de calcul alloué. On est ainsi assuré qu'aucun biais n'a été introduit par les jeux de données où il n'a pas été possible de calculer la solution exacte.

La modélisation par chaîne de Markov décrite en Section 4.2.5 converge vers la distribution uniforme lorsque le nombre d'étapes augmente. Nous avons estimé en Section 4.2.5 qu'un nombre de pas de 45998 était suffisant pour atteindre une distribution uniforme à $\varepsilon = 0.01$ près, arrondis ensuite à 50 000 pas. Expérimentalement, nous confirmons que ce nombre de pas permet d'atteindre une distribution uniforme par deux indicateurs.

Premièrement, la similarité moyenne de jeux de données générés uniformément est $s = -0.0388$ alors que les jeux de données générés par chaîne de Markov ont une similarité de $s = -0.0384$. On remarque en Figure 4.8 qu'avec 50 pas dans la chaîne de Markov la similarité est de $s = 0.88$. Deuxièmement, les résultats obtenus sur des jeux de données générés uniformément avec les mêmes nombres de classements et d'éléments sont équivalents aux résultats obtenus en utilisant des jeux de données générés avec la modélisation par chaîne de Markov.

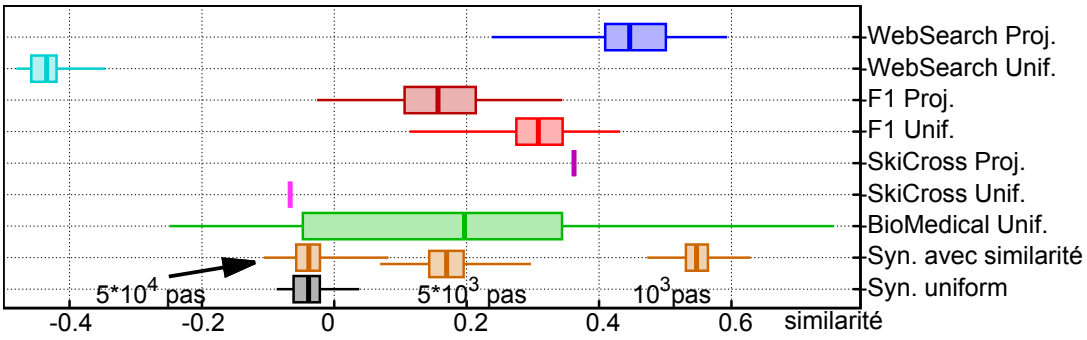

FIGURE 4.9 – Distribution de la similarité pour chaque groupe de jeux de données, groupé par type. Les jeux de données synthétiques avec similarité sont présentés dans trois configurations dépendants du nombre de pas utilisés lors de la génération par chaîne de Markov.

En Figure 4.9, nous présentons les similarités mesurées pour les jeux de données réels. On remarque que certains jeux de données, comme les WebSearch unifiés, ont une similarité homogène (de -0.48 à -0.34), alors que d'autres ont une similarité qui varie grandement. Dans le cas des jeux de données biomédicaux (BioMedical), la similarité varie de -0.25 à 0.75.

Observons maintenant l'impact de la similarité sur la production de consensus, à la fois en temps de calcul pour produire un consensus et sur la qualité des consensus produits.

4.3.2.1 Temps de calcul

Les expériences faites sur les jeux de données synthétiques avec différents niveaux de similarité concernant la consommation de temps révèlent un groupe d'algorithmes dont le temps de calcul diminue significativement lorsque la similarité intrinsèque des jeux de données augmente. Afin de mettre en évidence les algorithmes profitant de la similarité, nous présentons en Figure 4.10, pour chaque algorithme, le temps de calcul moyen obtenu pour un nombre de pas donné, divisé par le temps moyen le plus élevé pour n'importe quel nombre de pas par ce même algorithme.

Les algorithmes de recherche locale, tel que BIOCONSERT, devraient, de par leur conception, exploiter la similarité intrinsèque d'un jeu de données, d'autant plus que BIOCONSERT utilise comme point de départ un des classements fournis en entrée. Comparé aux jeux de données sans similarité (50 000 pas, $s = 0.038$), BIOCONSERT propose un consensus jusqu'à 57% plus rapide avec des jeux de données similaires (50 pas, $s = 0.88$). La plus grande influence observé est pour

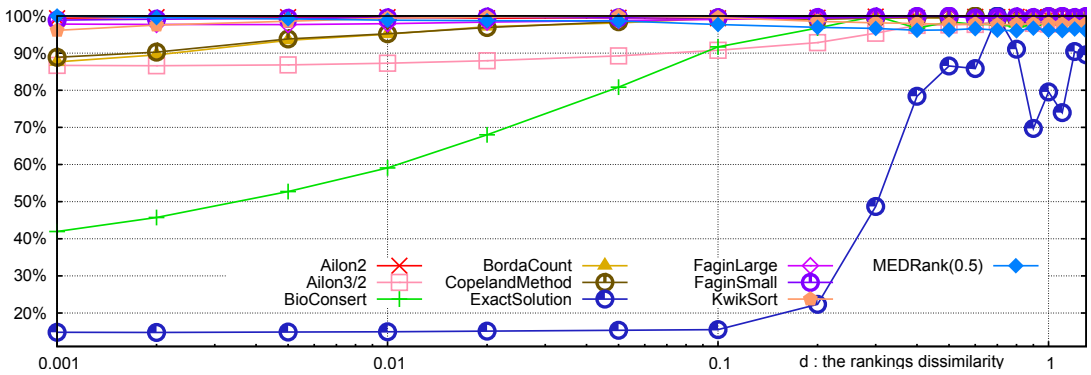

FIGURE 4.10 – Temps de calcul normé (ordonnée) pour des jeux de données synthétiques avec différents niveaux de similarité (nombre de pas en abscisse). Pour obtenir un temps de calcul normé, le temps de calcul moyen d'un algorithme pour un nombre de pas donnée est normé par le plus long des temps de calcul moyen obtenu pour un nombre de pas quelconque. En abscisse, on trouve le nombre de pas utilisé lors de la génération des données, et en ordonnée le temps de calcul normé.

l'algorithme ExactPNE qui propose des résultats 85% plus rapidement sur des jeux de données similaires. Les autres algorithmes tirent peu ou pas avantage de la similarité des jeux de données et ne sont pas détaillés ici.

4.3.2.2 Évaluation qualitative

Dans la Figure 4.11, nous présentons le *gap* moyen obtenu par les algorithmes pour différents nombres de pas lors de la génération des données, et donc différents niveaux de similarité (*cf.* Figure 4.8). Trois comportements intéressants ont été repérés dans la Figure 4.11. Premièrement, KwikSort est positivement influencé par la similarité, le *gap* moyen est 24 fois plus petit avec des jeux de données très similaires (50 étapes) par rapport aux jeux de données non-similaires (50 000 pas). BioConsert a un *gap* si réduit qu'on ne peut distinguer en Figure 4.11 s'il est influencé par la similarité. En observant ses résultats, on remarque qu'il trouve toujours la solution optimale des jeux de données similaires ($\leq$ 500 pas), et a un *gap* moyen de 0.02% avec les jeux de données non-similaires (50 000 pas). Les observations faites sur KwikSort sont confirmées avec l'utilisation de jeux de données réels : KwikSort est en effet plus efficace avec jeux de données similaires, mais il est aussi négativement affecté par la similarité négative des jeux de données *WebSearch Unifié* et *skicross Unifié* (*cf.* Figure 4.9).

Deuxièmement, BordaCount présente un *gap* très stable en Figure 4.11 pour les différents niveaux de similarité : le *gap* moyen varie de 3.6% à 4.0%. Éton-

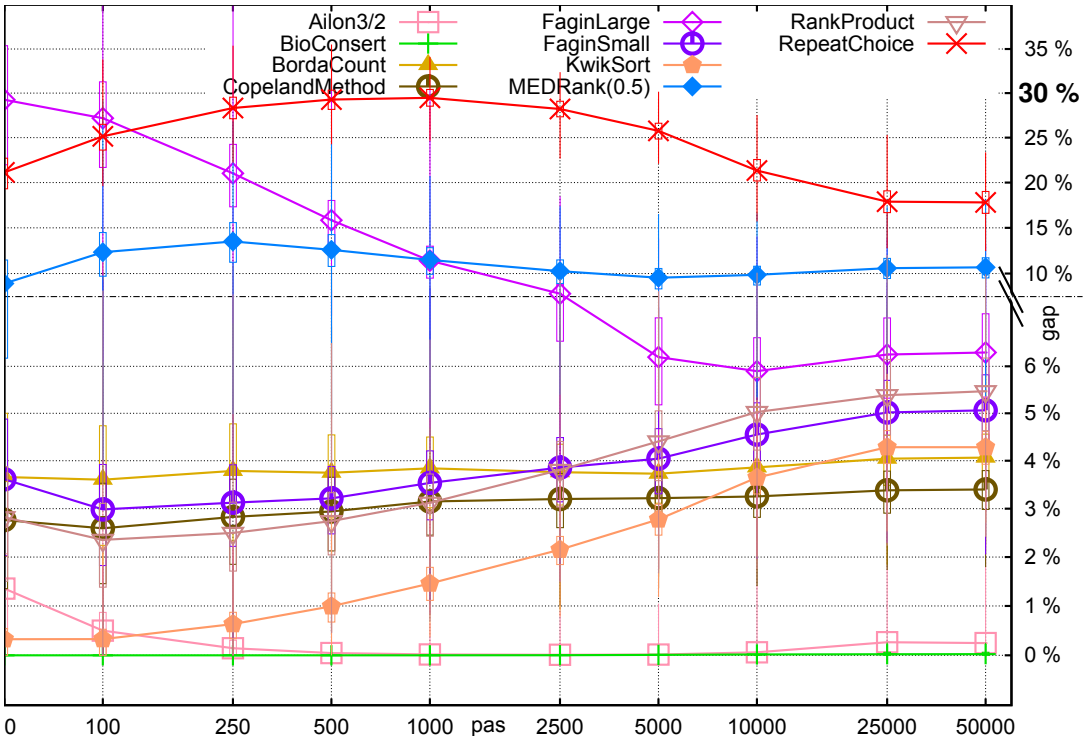

FIGURE 4.11 – En considérant les jeux de données synthétiques avec **similarité**, le *gap* est dessiné en ordonnée pour différents nombres de pas utilisés lors de la génération des données (*cf.* Section 4.2.5). On note le changement d'échelle pour un *gap* supérieur à 7.5% afin de distinguer les algorithmes ayant des *gap* faibles, et pouvoir visualiser les algorithmes ayant des *gap* élevés.

namment, cette stabilité ne peut être observée sur les jeux de données réels (*cf.* Table 4.3) : BORDACOUNT a un des plus élevés (et mauvais) *gap* sur les jeux de données de type *F1 unifié* (30.2%) alors qu'il est correct sur jeux de données de type *F1 projeté* (3.7%).

Troisièmement, FAGINLARGE est négativement influencé par la similarité des jeux de données synthétiques : le *gap* moyen est 4.7 fois plus grand avec des jeux de données très similaires (50 pas) qu'avec des jeux de données non similaires (50 000 pas). Encore une fois, cette observation ne peut être confirmée avec les jeux de données réels : alors que FAGINSMALL retourne de meilleurs résultats que FAGINLARGE pour les jeux de données *WebSearch projeté* qui sont similaires, c'est le contraire sur pour *F1 Unifié* alors qu'ils sont également similaires.

En conclusion, deux approches bénéficient particulièrement de la similarité en termes de qualité (Figure 4.11), à savoir BIOCONSERT et KWIKSORT, tandis que BIOCONSERT et EXACTPNE bénéficient de cette similarité quant à leur temps de

calcul respectifs.

Les observations faites sur la qualité de KwikSort (*cf.* Figure 4.11) ou le temps de calcul de BioConsert (*cf.* Section 4.3.2.1) montrent que la similarité est une propriété pertinente à connaître pour choisir entre différents algorithmes.

Les observations faites sur BordaCount ou FaginDyn avec des jeux de données synthétiques (*cf.* Figure 4.11), et qui ne peuvent être répétées sur des jeux de données réels (*cf.* Table 4.3 et Figure 4.9), mettent clairement en évidence que, pour certains algorithmes, connaître la taille d'un jeu de données ainsi que sa similarité intrinsèque ne suffissent pas pour faire un choix éclairé sur l'opportunité d'utiliser ces algorithmes. Pour cette raison, la section suivante se concentre sur l'influence du processus de normalisation dans la qualité des résultats.

4.3.3 Similarité et Normalisation

Dans cette sous section, nous nous intéressons a l'impact sur la qualité des consensus produit qu'ont deux processus de normalisation couramment utilisés, à savoir la projection et l'unification. Plus précisément, nous comparons en Section 4.3.3.1 ces deux processus de normalisation sur des jeux de données réels afin d'illustrer leur impact sur les jeux de données, puis étudions en Figure 4.3.3.2 l'influence du processus d'unification sur les résultats produits par les algorithmes.

4.3.3.1 Unification et projection de jeux de données réels

Lorsque l'on considère des jeux de données avec m classements et n éléments, on remarque que le processus de projection produit un jeu de données contenant de 0 à l éléments tandis que le processus d'unification produit un jeu de données contenant de l à $l \times m$ éléments.

Rappelons que les jeux de données *F1* sont des saisons du championnat de Formule 1, chaque classement est l'ordre d'arrivée des pilotes ayant fini la course. La projection a été appliquée par [BBN13] avec pour conséquence de supprimer $53.42\% \pm 25.03\%$ des pilotes. Parmi les pilotes retirés nous trouvons le vice-champion de 1961 et le champion de 1970. Ces deux pilotes peuvent clairement être qualifiés d'éléments *pertinents*, ils ne doivent donc pas être supprimés. En moyenne, les jeux de données projetés contiennent 15.81 ± 8.53 éléments alors que les jeux de données unifiés contiennent 38.73 ± 11.39 éléments.

Lorsque l'on considère les jeux de données *WebSearch*, la projection supprime en moyenne $98.42\% \pm 0.89\%$ des éléments (procédure suivie par [BBN13]). Chaque classement contient les mille meilleurs résultats pour un moteur de recherche donné ; la projection produit, en moyenne, des jeux de données contenant 40 ± 20

éléments alors que les jeux de données unifiés en contiennent 2586 ± 388 (utilisés dans [AM12 ; SZ09]).

Utiliser un processus de normalisation afin de faire le lien entre des jeux de données incomplets et des algorithmes calculant des consensus sur des jeux de données complets est un besoin crucial déjà mis en avant au chapitre précédent. La projection est un processus de normalisation qui peut supprimer des éléments pertinents, tels que le champion dans les jeux de données F1. Le processus d'unification peut, quant à lui, augmenter radicalement le nombre d'éléments dans chaque classement d'un jeu de données lorsqu'il y a peu de recouvrement entre les classements (il peut y avoir de 2.4 à 65 fois plus d'éléments avec l'unification sur les jeux de données réels utilisé dans ce chapitre). Le choix entre ces deux processus doit donc être un choix averti, et le guide à l'utilisateur introduit au chapitre précédent permet de faire ce choix.

On note finalement qu'avoir une grande différence de taille entre les jeux de données projetés et unifiés est un indicateur de la présence de grands groupes d'unification. À titre d'exemple, dans les jeux de données *WebSearch*, la taille moyenne des groupes d'unification est de 1586.

Nous étudions ci-après ces éventuellement grands groupes d'unification plus en détail, ainsi que l'impact qu'ils ont sur la qualité des résultats.

4.3.3.2 Impact de l'unification sur des jeux de données similaires

Plus le nombre d'éléments en commun entre les classements en entrée est petit avant d'appliquer le processus d'unification, plus la taille des groupes d'unification est grande. Sur les jeux de données synthétiques unifiés avec similarités (défini en Section 4.2.6), la taille moyenne des groupes de jeux de données similaires (générés en 10^3 étapes) est 1.52 alors qu'il est 6.52 pour des jeux de données non-similaires (générés en 10^6 étapes). Cette augmentation de la taille des groupes devrait permettre de séparer les algorithmes en deux catégories selon qu'ils prennent en compte ou non le coût de faire/rompre des égalités. La taille moyenne des groupes d'égalités et les *gap* des algorithmes est présenté en Figure 4.12.

D'une part BioConsert, KwikSort et MEDRank considèrent l'importance de créer et rompre des égalités, et sont donc susceptibles d'être stables en termes de qualité, une hypothèse confirmée dans cette expérimentation.

D'autre part, BordaCount, CopelandMethod et RepeatChoice ne sont pas en mesure de prendre en compte le coût de créer/rompre des égalités. Ces algorithmes devraient être considérablement influencés par le processus d'unification qui peut créer de grands groupes d'unification. Expérimentalement, ils induisent 15 fois plus de désaccords avec des jeux de données non-similaires et unifiés qu'avec des jeux de données similaires et unifiés. La variation de la qualité des résultats

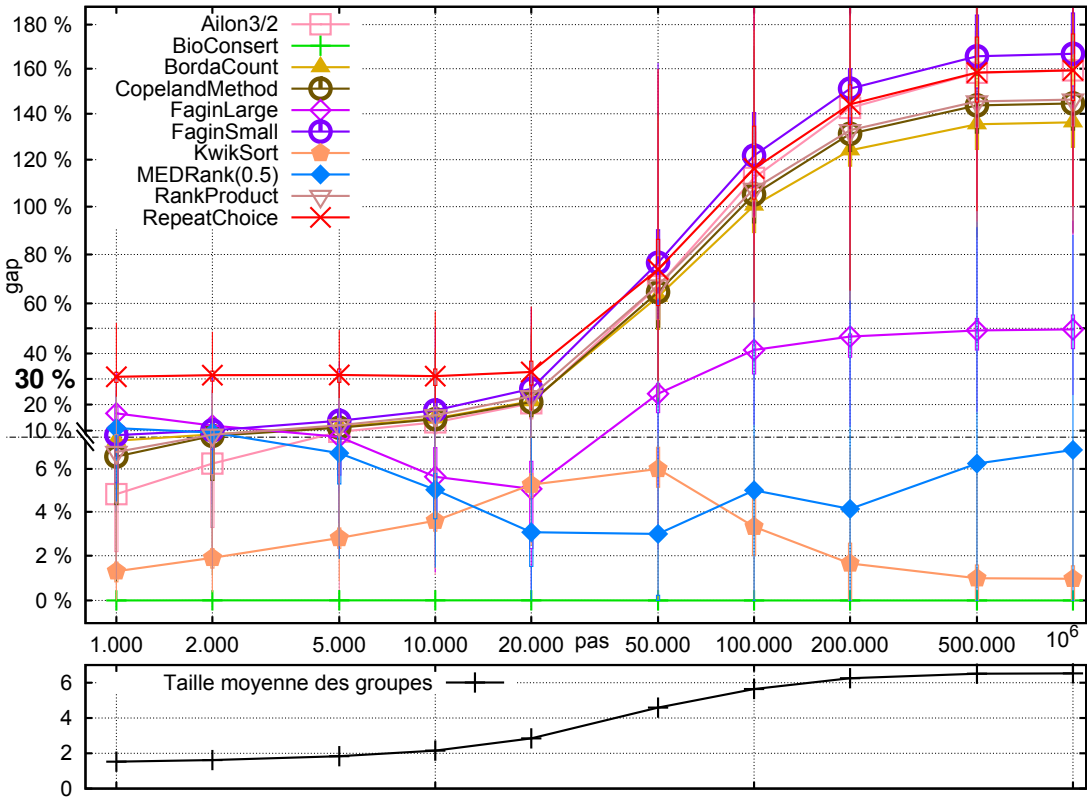

FIGURE 4.12 – Valeurs de *gap* (graphique supérieur) et taille des groupes (graphique inférieur) pour différents nombres de pas utilisés lors de la génération de données synthétiques **unifiés** (*cf.* Section 4.2.6). Dans un souci de comparaison avec la Figure 4.11, on applique le même changement d'échelle pour les valeurs de *gap* supérieures à 7.5%.

pour BORDACOUNT et COPELANDMETHOD avec les jeux de données réels qui n'étaient jusque-là pas reproductibles avec des jeux de données synthétiques, qu'ils soient similaires ou non, est maintenant entièrement reproduite (cf. Figure 4.12 vs Figure 4.11).

Les variantes de FAGINDYN ont des comportements qui ne sont pas cohérents entre les jeux de données réels et synthétiques lorsqu'on ne prend en compte que la taille et la similarité des jeux de données (*cf.* Section 4.3.2). Dans cette expérience, on observe clairement que le processus d'unification, conduisant à la création d'un grand groupe d'éléments à la fin des jeux de données non-similaires, a un impact négatif sur FAGINSMALL. Dans cette expérience, favoriser des petits groupes d'éléments à égalité alors qu'il y a de grands groupes dans les classements d'entrée est clairement un choix désavantageux pour FAGINSMALL comme cela apparaît en

Figure 4.12.

À la fois sur les jeux de données unifiés synthétiques et réels, CopelandMethod calcule des consensus de meilleure qualité que BordaCount. Il est à noter que la qualité des consensus produits pour les jeux de données projetés est la même pour ces deux algorithmes.

Le processus d'unification est une approche de normalisation qui induit la présence de grands groupes d'éléments à égalité. Elle est maintenant connue comme étant la cause de la mauvaise performance de BordaCount et FaginSmall. Les résultats obtenus sur des jeux de données réels et synthétiques sont compatibles les uns avec les autres, maintenant que l'impact du processus de normalisation est compris.

4.3.3.3 Discussion sur les égalités dans les données synthétiques uniformément générées

Un comportement surprenant a été observé en étudiant BordaCount et CopelandMethod sur les données synthétiques générés uniformément : leurs résultats, comparés aux autres, s'améliorent lorsque le nombre d'éléments dans les jeux de données augmente. Ce comportement n'a pu être expliqué en ne considérant que la taille des jeux de données. En travaillant avec les données synthétiques unifiées, nous avons remarqué que BordaCount et CopelandMethod proposent des résultats qui se dégradent lorsqu'il y a beaucoup d'égalités entre éléments dans les jeux de données.

Étudions maintenant les données synthétiques générés uniformément, et plus particulièrement les égalités présentes dans ces jeux de données. Dans un premier temps, nous justifions de façon théorique que l'amélioration relative de BordaCount et CopelandMethod est due à la présence, de plus en plus faible, d'égalités entre éléments à mesure que le nombre d'éléments augmente. Dans un deuxième temps nous étudions expérimentalement les égalités dans les jeux de données et les consensus associés.

Considérons deux éléments A et B, le nombre de classements dans $\mathcal{R}_n$ où ils sont à égalité est $|\mathcal{R}_{n-1}|$, la probabilité qu'ils soient ensemble est alors $\frac{|\mathcal{R}_{n-1}|}{|\mathcal{R}_n|}$. En effet, les classements ici considérés ont été générés de façon uniforme, c'est-à-dire que chaque classement a la même probabilité d'être tiré. La probabilité qu'une quelconque paire d'éléments soit à égalité est alors de $\frac{n(n-1)}{2} * \frac{|\mathcal{R}_{n-1}|}{|\mathcal{R}_n|}$. Le nombre de paires d'éléments dans un classement de $\mathcal{R}_n$ croit polynomialement avec n, mais le nombre de classements croit lui exponentiellement plus vite que $n!$ [Slo+03]. La probabilité d'avoir deux éléments à égalité est donc décroissante en fonction de n le nombre d'éléments. On conclut donc que plus on génère des classements avec un grand nombre d'éléments, plus la probabilité que deux éléments soient à égalité

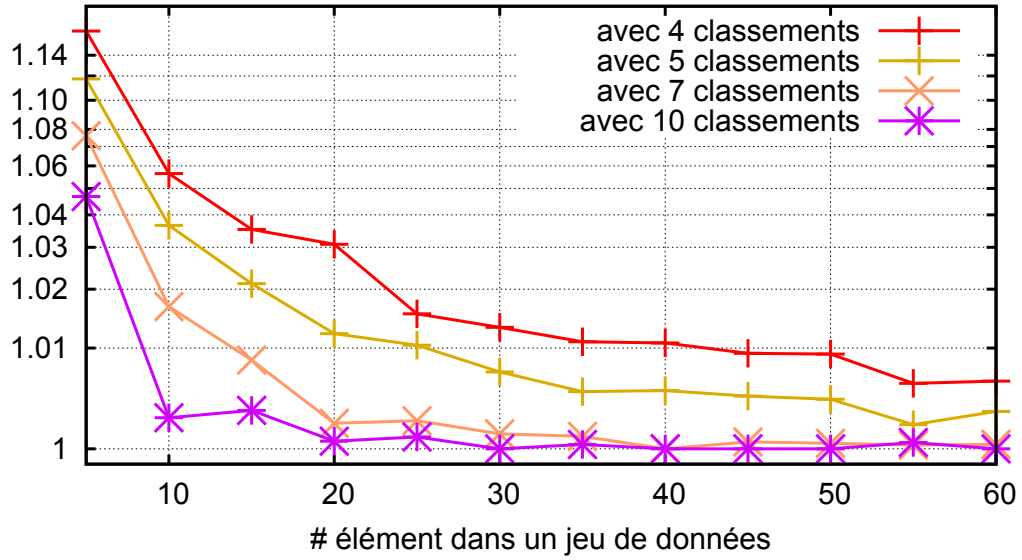

FIGURE 4.13 – Nombre moyen d'éléments dans les consensus optimaux calculés pour les jeux de données synthétiques uniformément générés avec différents nombres de classements par jeux de données (4, 5, 7 ou 10).

dans plusieurs classements diminue, et donc a fortiori dans le consensus. De même la probabilité que deux éléments soient à égalité dans m classements est plus forte que lorsque l'on considère $m + 1$ classements. On en déduit que plus le nombre de classements ou d'éléments augments, moins il y aura d'égalités entre éléments dans les consensus.

Observons maintenant comment évolue la taille moyenne des groupes dans les consensus optimaux trouvés (liée au nombre d'égalités dans ces consensus) par rapport au nombre d'éléments et de classements dans les jeux de données. Nous présentons en Figure 4.13 l'évolution de la taille moyenne des groupes dans les consensus optimaux des jeux de données contenant 4, 5, 7 ou 10 classements pour différents nombres d'éléments. On remarque en effet que la taille moyenne des groupes diminue à mesure que le nombre d'éléments augmente, mais aussi que plus on augmente le nombre de classements dans un jeu de données plus le nombre d'égalités dans le consensus diminue.

Nous avons remarqué en Section 4.3.1.1 que BORDACOUNT et COPELANDMETHOD pouvaient à tort rompre des égalités. Nous avons mis en évidence qu'il y a moins d'égalité dans les consensus optimaux des jeux de données générés uniformément lorsque le nombre d'éléments et de classements augmente. Nous concluons donc qu'il est cohérent que des algorithmes comme BORDACOUNT et COPELANDMETHOD proposent de meilleurs résultats lorsque le nombre d'éléments augmente.

4.4 Recommandations en fonction des caractéristiques des jeux de données et des besoins des utilisateurs

En se basant sur les expériences conduites dans la section précédente, nous sommes en mesure de fournir des recommandations dans le choix d'un algorithme en fonction des propriétés connues des jeux de données à traiter et du compromis souhaité entre l'efficacité du temps et de la qualité attendue des résultats. Alors que la Section 3.2 a présenté un *guide à l'utilisateur* dans le choix de la distance ou mesure et du processus de normalisation à utiliser en fonction de ses besoins, nous présentons ici un second *guide à l'utilisateur* dans le choix d'un algorithme.

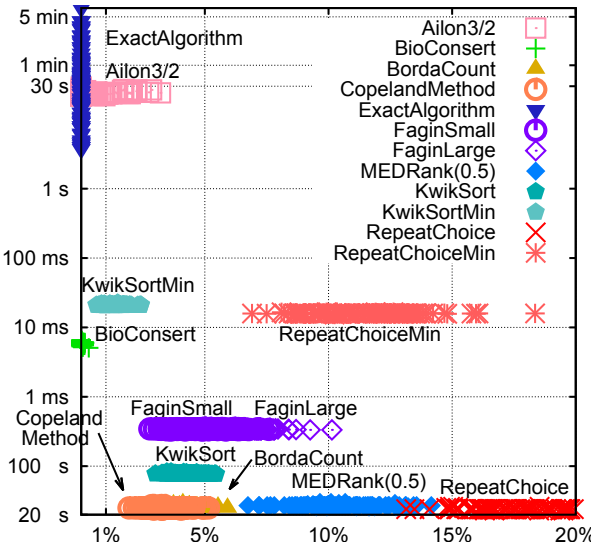

FIGURE 4.14 – Temps de calcul et *gap* obtenu par des algorithmes pour des jeux de données générés uniformément avec $m = 7$ classements contenant $n = 35$ éléments

La Figure 4.14 illustre le choix qui pourrait être proposé à l'utilisateur entre les différents algorithmes. Il a été produit avec 100 jeux de données générés uniformément avec $m = 7$ classements contenant $n = 35$ éléments. Les résultats sont représentatifs d'autres jeux de données générés de manière uniforme.

La conclusion générale est que BIOCONSERT est la meilleure approche à suivre dans un très grand nombre de cas. Tout en étant raisonnablement plus lent que d'autres algorithmes, BIOCONSERT profite de la similarité des jeux de données, et est indépendant du processus de normalisation appliquée pour produire des résultats de qualité. Dans des situations extrêmes, des alternatives peuvent être envisagées.

Premièrement, lorsque des résultats d'un qualité supérieur à BIOCONSERT sont demandés, l'algorithme EXACTPNE doit être envisagé, et ce malgré son temps de calcul exponentiel.

Deuxièmement, lorsque de très grands jeux de données sont considérés (avec un nombre d'éléments $n > 30\ 000$), l'implémentation actuelle de BIOCONSERT avec une complexité de mémoire $\mathcal{O}(n^2)$ peut faire face à des limitations physiques, et

peut en outre prendre du temps pour proposer un consensus. Dans ce cas KwikSort est alors la meilleure alternative pour les résultats de bonne qualité, d'autant plus qu'il est positivement influencé par la similarité des jeux de données (*cf.* Figure 4.11).

Finalement, si le temps est très important, alors BordaCount doit être considéré, mais uniquement lorsqu'il y a peu d'égalités dans les classements. Lorsqu'il y a beaucoup d'égalités entre les classements (éventuellement obtenus par le processus d'unification) MEDRank est un excellent candidat.

4.5 Réutilisation, reproductibilité et diffusion

Réutiliser les algorithmes et les jeux de données de la littérature a été une des problématiques techniques majeures de ce travail [Bra+15]. En effet les jeux de données utilisés par les études précédentes ne sont pas tous disponibles, ni sous les mêmes formats (il n'existe à notre connaissance aucun standard pour décrire des jeux de données avec égalités). Les implémentations des algorithmes ne sont quant à elles que très rarement disponibles. Afin de poursuivre le but de ce travail, nous avons proposé la plateforme Rank'n'ties, disponible à l'adresse `http://rank-aggregation-with-ties.lri.fr/`. Plus précisément, les contributions de Rank'n'ties peuvent être décrites en autour de quatre axes. Premièrement, cette plateforme permet de reproduire les analyses car elle contient l'ensemble des résultats produits. Deuxièmement, la plateforme permet l'ajout de nouveaux algorithmes afin de les comparer aux résultats des autres algorithmes déjà enregistré en base de données. Troisièmement, Rank'n'ties permet l'ajout de nouveaux jeux de données et de lancer de nouveaux calculs sur ces derniers. Finalement, Rank'n'ties permet de télécharger l'ensemble des jeux de données avec une documentation détaillée du format de données utilisé.

Rank'n'ties a été principalement réalisé par Adrien Bestel, dans le cadre d'un stage de $4^{ième}$ année d'ingénieur (3 mois), encadré par Sylvie Hamel (Professeur à l'Université de Montréal) et moi-même. J'ai ensuite finalisé l'implémentation de Rank'n'ties et amélioré son passage à l'échelle.

Cette section s'organise de la façon suivante. Premièrement nous décrivons l'interface de Rank'n'ties. Ensuite, nous présentons les fonctionnalités principales de la plateforme, à savoir les différentes évaluations réalisables sur les données, mais aussi la possibilité d'ajouter des données, et surtout de nouveau algorithmes. Dans un troisième temps, nous détaillerons les technologies mises en oeuvre dans cette plateforme. Finalement, nous illustrerons cette plateforme avec plusieurs cas d'utilisation.

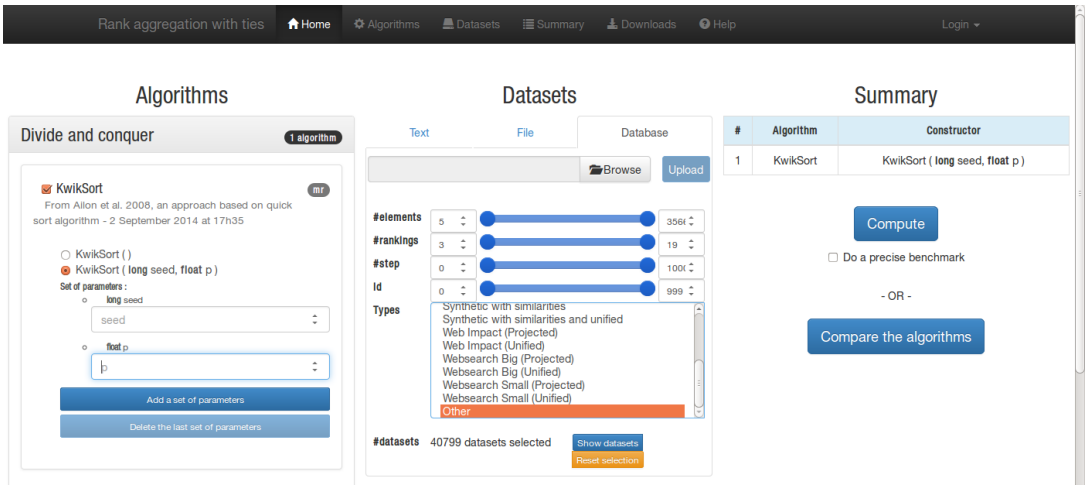

FIGURE 4.15 – Interface principale de Rank'n'ties. Dans la colonne de gauche, l'utilisateur choisit les algorithmes à évaluer. Dans la colonne centrale, les données sur lesquelles évaluer ces algorithmes. Dans la colonne de droite l'utilisateur indique s'il veut visualiser les résultats déjà présents dans la base de donnée, ou en produire de nouveaux.

4.5.1 Interfaces principales de l'outil

La plateforme Rank'n'ties se décompose en quatre interfaces : l'interface où l'on choisit les calculs à faire (*cf.* Section 4.5.1.1), celle où l'on observe les résultats (*cf.* Section 4.5.1.2), celle où l'on observe les résultats lorsque l'on ne considère que quelques jeux de données (*cf.* Section 4.5.1.3) et l'interface de gestion de compte et des résultats (*cf.* Section 4.5.1.4).

4.5.1.1 Interface "Choix"

L'interface principale de Rank'n'ties est présentée en Figure 4.15 et permet de choisir les algorithmes et les données sur lesquelles lancer des calculs. Une option permet de mesurer de façon précise le temps nécessaire que met chaque algorithme pour calculer chaque consensus. L'interface permet, lors du choix des algorithmes, de spécifier des valeurs pour leurs paramètres. Par exemple, la Figure 4.15 illustre le choix pour KWIKSORT de la valeur de p pour la distance de Kendall-τ généralisée et de la graine utilisée dans le générateur aléatoire de l'algorithme. Cette interface principale permet aussi d'ajouter de nouveaux algorithmes (bouton en bas de la colonne de gauche), et des jeux de données personnalisés (colonne du milieu via "Browse" et "Upload").

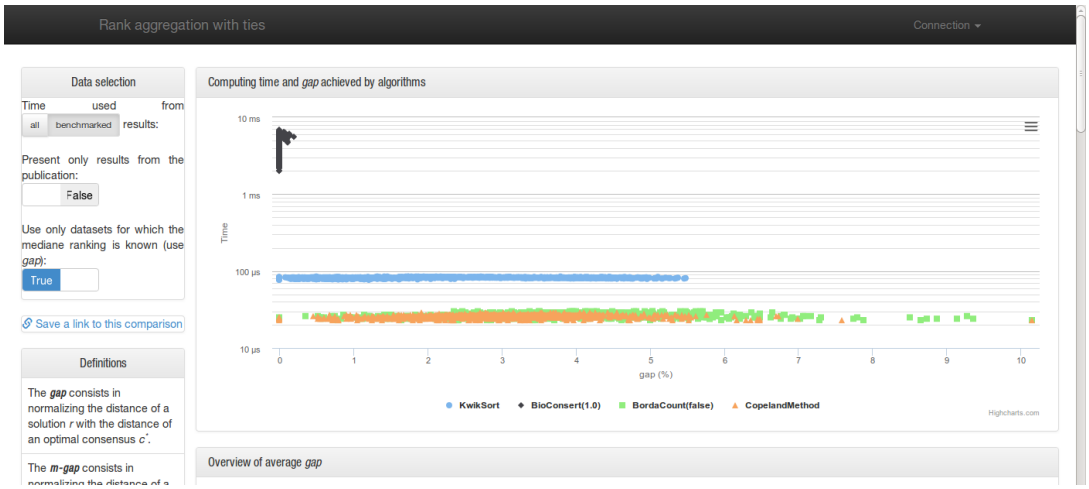

FIGURE 4.16 – Interface présentant les résultats lorsque de nombreux jeux de données ont été sélectionnés, dans ce cas on agrège les résultats dans plusieurs graphiques.

4.5.1.2 Interface "Résultats en grand nombre"

La seconde interface est celle de la présentation des résultats. Lorsque l'utilisateur a lancé des calculs sur un grand nombre de données, les résultats sont présentés via plusieurs graphiques comme on le voit en Figure 4.16 avec entre autres une présentation des résultats similaires à la Figure 4.14. Les graphiques présentés à l'utilisateur seront détaillés dans la sous-section suivante.

4.5.1.3 Interface "Résultats en détails"

Lorsque les calculs ont été lancés sur un nombre réduit de jeux de données (actuellement 10), Rank'n'ties présente les consensus calculés par chaque algorithme. On observe en Figure 4.17 que Rank'n'ties présente ainsi les temps de calcul et la distance des consensus au jeu de données et le consensus, et finalement permet de télécharger ces résultats.

4.5.1.4 Interface "Compte et données"

Finalement, l'interface de gestion de compte (*cf.* Figure 4.18) permet d'accéder aux résultats de l'ensemble des calculs lancés, de les télécharger au format JSON (un format de fichier standard basé sur une organisation arborescente de l'information). Cette interface permet aussi d'accéder aux algorithmes ajoutés dans la plateforme et de les télécharger depuis cette dernière, mais aussi les supprimer.

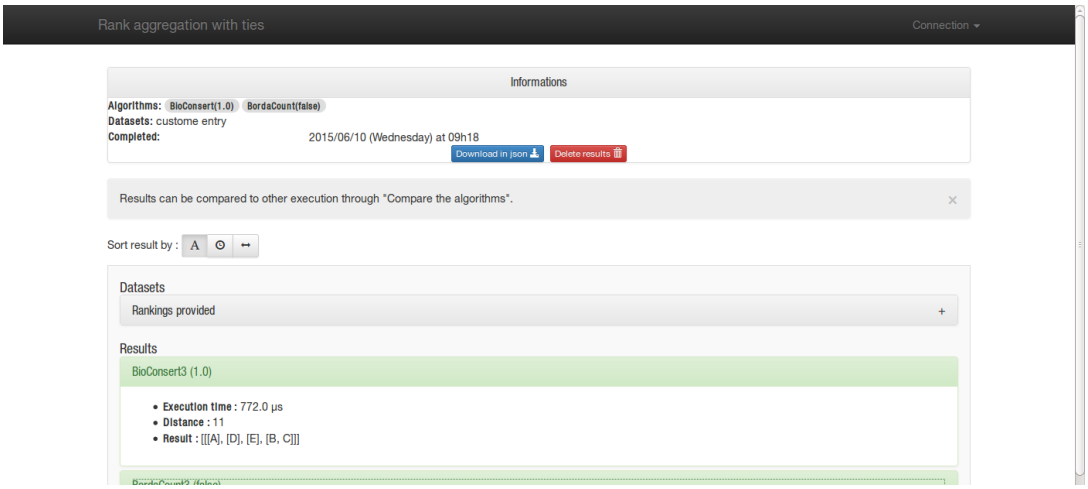

FIGURE 4.17 – Interface présentant les résultats lorsque peu de jeux de données on été sélectionnés, dans ce cas on affiche les consensus calculés.

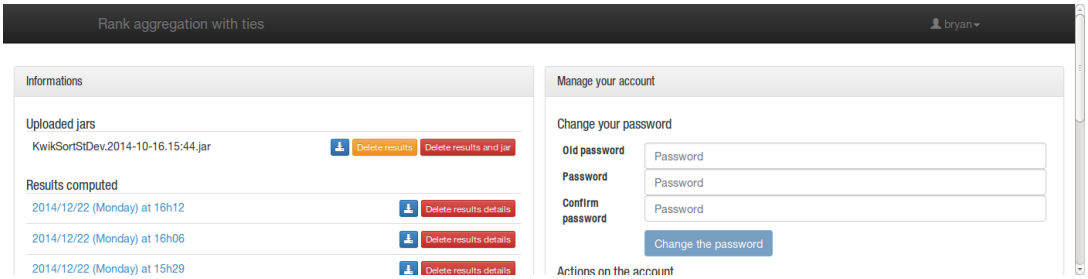

FIGURE 4.18 – Gestion des calculs lancés et du compte, on peut y télécharger des résultats, des algorithmes précédemment envoyés, mais aussi modifier/supprimer son compte.

4.5.2 Visualisation des résultats

Le but de cet outil est de rendre accessible tous les outils qui ont permis les analyses faites dans [Bra+15], à savoir d'évaluer les performances en temps et en qualité des différents algorithmes pour des données avec diverses caractéristiques. Afin de s'inscrire dans une perspective d'extension des analyses précédemment produites, nous proposons d'évaluer les résultats au travers de graphiques et figures déjà utilisés dans de précédents travaux [AM12 ; BBN13 ; CDK06 ; SZ09].

4.5.2.1 Comparaison des résultats

Lorsque les calculs ont été effectués avec l'option permettant de mesurer précisément le temps nécessaire pour le calcul de chaque consensus par chaque algorithme,

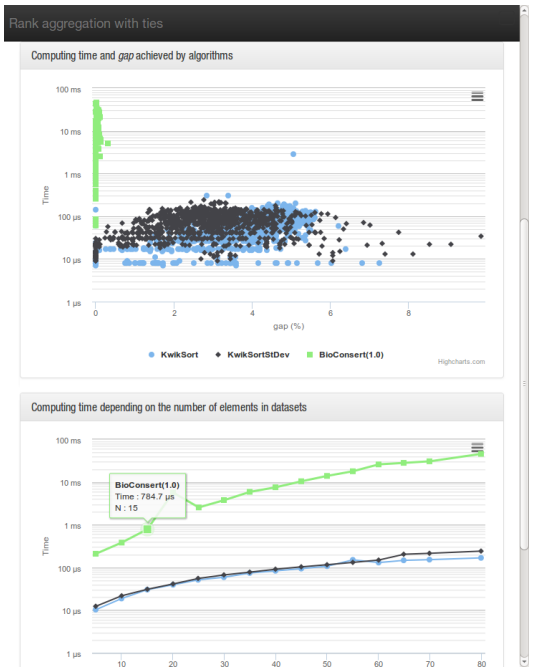

FIGURE 4.19 – Le graphique supérieur présente les résultats avec en abscisse le *gap* et en ordonnée le temps nécessaire pour calculer le consensus.Le graphique inférieur présente les temps de calculs moyen pour différent nombre d'éléments dans le jeu de données.

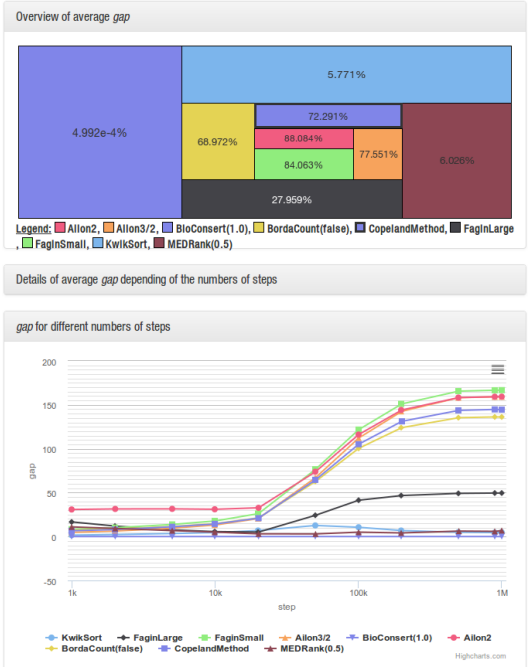

FIGURE 4.20 – Le graphique supérieur présente les *gap* moyens sous forme d'un *treemap ordonné*. Le graphique inférieur présente les *gap* moyens en fonction du nombre de pas utilisé lors de la génération des données.

le temps est exploité dans deux graphiques qui sont présentés en Figure 4.19. Dans le graphique supérieur, chaque point représente un algorithme pour un jeu de données, l'abscisse indiquant la qualité du résultat obtenu (via le *gap*), et l'ordonnée le temps nécessaire pour obtenir le consensus. Dans ce graphique, l'algorithme en noir est KwikSortStDev, une variante de KwikSort qui lui est en bleu. Cette variante semble proposer de meilleurs résultats que la version classique mais au prix d'une consommation en temps légèrement plus élevé. Ces deux algorithmes étant à la fois moins coûteux, mais moins bons que BioConsert (vert).

La partie inférieure de la Figure 4.19 est un graphique plus classique avec en abscisse, le nombre d'éléments dans les jeux de données et en ordonnée le temps de calcul moyen pour chaque algorithme (échelle logarithmique). On remarque ici aussi que KwikSortStDev est plus lent que KwikSort, mais de peu, et que BioConsert est bien plus coûteux en temps que les deux autres.

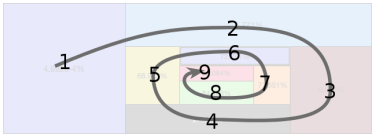

FIGURE 4.21 – Les algorithmes occupent la surface allouée à la visualisation selon une spirale tournant dans le sens horaire de l'extérieur vers l'intérieur.

Afin de comparer la qualité moyenne des résultats, deux types de visualisation sont proposés. Le premier type de visualisation est basé sur un *treemap ordonné* (*Ordered TreeMap*) [SW01 ; TS07] visible en haut de la Figure 4.20. Le second type est visible dans la partie inférieure de Figure 4.20, on y trace le *gap* en ordonnée. Concentrons-nous dans un premier temps sur le *treemap ordonné*.

En Figure 4.20, le schéma du haut est un *treemap* ordonné où l'on représente une visualisation du *gap* moyen des consensus de chaque algorithme. Dans cette visualisation, les algorithmes de consensus sont triés par leur *gap*, puis placés selon une spirale allant de l'extérieur vers l'intérieur dans un sens horaire en spirale (*cf.* Figure 4.21). Plus précisément, le meilleur algorithme est représenté dans le tiers ouest de la visualisation, le second recevant le tiers nord de ce qui est encore disponible, le troisième recevant le tiers est de ce qui est disponible, le quatrième le tiers sud et ainsi de suite. Lorsqu'il y a égalité entre deux algorithmes, ils se partagent 40% de l'espace disponible. À noter que l'avant-dernier algorithme reçoit 60% de l'espace disponible, et le dernier ce qu'il reste.

Afin de mettre en évidence des variations de la qualité des résultats liés à divers paramètres, on représente le *gap* en fonction de différentes abscisses. Si les données sont synthétiques et générées avec la modélisation par chaîne de Markov (*cf.* Section 4.2.5) (unifiée ou non), on représente le *gap* en fonction du nombre de pas parcourus dans la chaîne de Markov lors de la génération. Le graphique présenté en Figure 4.20 reproduit la Figure 4.12 et y représente le *gap* pour des pas allant de 10^3 à 10^6. Pour les jeux de données synthétiques uniformément générés, mais aussi les jeux de données réels, ce graphique est produit en prenant comme abscisse le nombre d'éléments présents dans les jeux de données.

4.5.2.2 Comparaison sur de nouveaux jeux de données

Trois approches peuvent être suivies pour exécuter/évaluer des algorithmes sur de nouveaux jeux de données ou obtenir des consensus sur ces jeux de données. Pour cela, la partie centrale de la Figure 4.15, nommé *"Datasets"*, propose trois onglets. L'onglet "Text" permet de saisir manuellement un jeu de données. L'onglet "File" permet d'envoyer un fichier texte dans lequelle on a préalablement écrit son jeu de données. Finalement, l'onglet "Database" permet d'envoyer plusieurs fichier et le sauvegarder dans la base de données de Rank'n'ties afin de s'en servir ultérieurement. Il suffit ensuite de sélectionner le type de jeux de données (*datasets*) "Others".

4.5.2.3 Comparaison de nouveaux algorithmes par rapport à l'existant

Dans l'interface principale présentée en Figure 4.15, la colonne de gauche permet de choisir parmi différents algorithmes. Afin d'évaluer de nouvelles approches, tout utilisateur peut envoyer dans Rank'n'ties un algorithme et ensuite l'évaluer. Pour des raisons évidentes de confidentialité, cet algorithme n'est visible que par l'utilisateur qui l'a changé. Pour envoyer et évaluer un nouvel algorithme, l'utilisateur doit dans un premier temps télécharger une archive contenant un squelette d'algorithme d'agrégation de classements (détaillé dans la sous section suivante), à charge de l'utilisateur de faire le lien entre son algorithme et le squelette. Ce dernier pouvant tout autant écrire l'algorithme en Java, ou l'écrire dans un autre langage et appeler son code depuis le squelette Java.

4.5.3 Squelette pour concevoir et implémenter de nouveaux algorithmes

La plateforme Rank'n'ties permet à chaque utilisateur d'envoyer son algorithme dans Rank'n'ties pour ensuite l'évaluer et le comparer à d'autres algorithmes. La plateforme permet, de plus, à l'utilisateur d'évaluer son algorithme avec plusieurs paramétrages. On peut aussi adjoindre à chaque algorithme et paramètres des commentaires pour permettre utilisation une plus simple. Dans cette optique, tous les algorithmes évalués dans [Bra+15] et présents dans Rank'n'ties se sont vus ajouter des commentaires pour expliquer leur paramètres. Toutes ces possibilités ont été mises en œuvre grâce à l'utilisation du langage Java.

Pour envoyer et évaluer un nouvel algorithme, un utilisateur doit *hériter* d'une *interface* précise, ce qui en Java correspond à respecter un cahier des charges en termes de présence de fonction et de noms des fonctions. Il doit par exemple posséder une fonction prenant en argument des classements et retournant un consensus, la fonction `computMedianRanking` en Figure 4.22.

Afin de fournir des commentaires sur les fonctions et paramètres de son algorithme, un utilisateur peut utiliser les *annotations*. En Figure 4.22 on indique ainsi que le paramètre `itm` peut avoir un nom plus explicite `IterationMax`, et qu'il contrôle en fait le nombre maximum d'itération. À noter que ces annotations permettent aussi de rendre visibles ou invisibles certaines fonctions et constructeurs.

Pour faciliter la prise en main de tous ces outils par l'utilisateur, nous avons mis à disposition une archive d'un projet Eclipse [1] contenant à la fois un exemple d'un algorithme, mais aussi un algorithme "coquille vide" dans lequel en écrire un nouveau.

1. Le logiciel Eclipse est un *Environnement de développement intégré* permettant la création de programme en java.

```
@MRAnnotation(description = "Morceau de classe présentant l'héritage et les annotations",
              name = "Ma classe exemple",
              virtualPackage = "Recherche locale")
public class AlgorithmeBasic<T> implements MedianRanking<T> {

    @MRConstructorAnnotation(hide = false)
    public ExampleAnnotation(@MRParameterAnnotation(defaultValue = "1.0",
                                                    description = "Nombre maximum d'ité
                                                        ration durant le recherche d'un
                                                        consensus",
                                                    name = "IterationMax")
                                                    int itm) {}

    @Override
    public List<Collection<T>> computMedianRanking(Collection<List<Collection<T>>>
        rankings) {
        return null;
    }
    (...)
}
```

FIGURE 4.22 – Extrait de code illustrant l'héritage et les annotations à utiliser lors de la réalisation d'un nouvel algorithme en vue de l'évaluer dans Rank'n'ties.

4.5.4 Informations techniques

Si on devait décrire Rank'n'ties par seulement deux fonctionnalités, ce serait sa capacité à dessiner des graphiques de différente natures, mais surtout la possibilité donnée à l'utilisateur d'y tester ses propres algorithmes.

Cette facilité à utiliser d'autres algorithmes a été rendu possible par l'utilisation des annotations dans le langage Java, ces dernières permettent d'écrire des commentaires textuels pour chaque classe, méthode et paramètre de méthode durant l'exécution. Pouvoir trouver les algorithmes envoyés par l'utilisateur dans des archives jar a été rendu possible par l'héritage, et par le fait que le langage autorise de lire les classes et les étudier comme n'importe quel objet.

Les visualisations graphiques sont basées sur l'utilisation de la librairie JavaScript Highcharts (`http://www.highcharts.com/`). Cette bibliothèque permet outre le dessin de nombreux graphiques, de les exporter dans divers formats. La visualisation en Treemap a été spécialement conçue par moi-même dans le cadre de [Bra+15] et de la plateforme Rank'n'ties.

Finalement, l'aspect général de la plateforme est basé sur la bibliothèque de fonction *Twitter Bootstrap* qui fournit un cadre de travail HTML/CSS riche permettant de s'adapter facilement tant au large écran qu'au téléphone connecté et est supporté par les principaux navigateurs, ce qui assure à Rank'n'ties une expérience homogène entre tous les utilisateurs.

4.5.5 Exemples d'utilisation

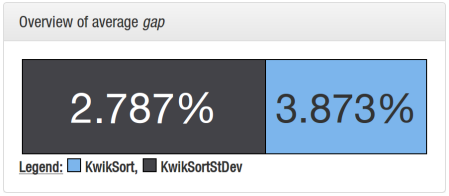

FIGURE 4.23 – TreeMap ordonné

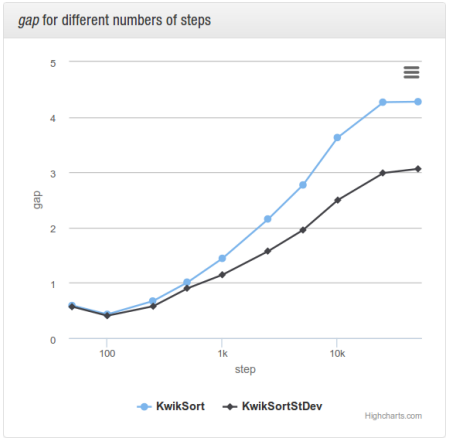

FIGURE 4.24 – Export pour le graphique *gap* pour les données similaires

Nous illustrons Rank'n'ties en suivant un scénario où un utilisateur veut évaluer une variante d'un algorithme à l'aide de la plateforme.

Considérons que l'utilisateur s'est penché sur l'algorithme KWIKSORT. Ce dernier est un algorithme récursif basé sur le choix d'un pivot pour diviser les autres éléments en trois groupes, ceux qui seront avant le pivot dans le consensus final, ceux qui seront après le pivot, et ceux qui seront à égalité avec le pivot. Nous avons vu précédemment que le choix du pivot influe grandement sur la qualité du consensus produit. L'utilisateur veut alors comparer KWIKSORT à une variante deterministe où le pivot est choisie en fonction d'un critère précis. Ce critère est l'écart-type de la position d'un élément dans le jeu de données par ordre décroissant, et la variante se nomme alors KWIKSORTSTDEV.

Une première analyse sur les données synthétiques générées uniformément présenté en Figure 4.19 permet de voir que KWIKSORTSTDEV est légèrement plus lent et semble proposer des résultats de meilleure qualité. En Figure 4.23, la visualisation par TreeMap permet de confirmer que KWIKSORTSTDEV produit de meilleurs résultats, et de visualiser le *gap* moyen.

Une seconde analyse peut ensuite être conduite sur les données générées avec différents niveaux de similarité, la visualisation du *gap* moyen en fonction du nombre de pas est présenté en Figure 4.24. On remarque alors que l'avantage de KWIKSORTSTDEV est lié au fait que les données ne soient pas similaires. Ce comportement est lui aussi observé sur les données synthétiques unifiées.

De ces analyses, on conclut que la variante KWIKSORTSTDEV permet d'améliorer KWIKSORT de façon significative et à moindre coût en réduisant l'impact négatif qu'a la non-similarité des jeux de données sur KWIKSORT.

4.6 Conclusion

Dans ce chapitre, nous avons évalué plusieurs algorithmes que nous avons introduits ou adaptés pour calculer des consensus entre classements, et ce en prenant en compte les égalités entre éléments. De ces évaluations, nous avons pu extraire un ensemble de recommandations permettant aux utilisateurs de ces algorithmes de choisir lequel utiliser en fonction de leurs besoins et de la nature des jeux de données.

Au cours des analyses, nous avons pu observer la stabilité et la qualité des consensus calculés par BioConsert. Nous avons pu observer la rapidité à laquelle les algorithmes positionnels peuvent produire des consensus d'une qualité acceptable, mais aussi l'impact très négatif que peut avoir le processus d'unification et/ou la présence de grands groupes d'éléments à égalité sur la qualité des résultats de BordaCount et CopelandMethod. Les analyses nous ont finalement permis d'observer que KwikSort offrait un bon compromis qualité/temps.

Nous avons finalement présenté Rank'n'ties, une plateforme permettant de reproduire et étendre les analyses via de nouveaux algorithmes et jeux de données.

De ces expérimentations apparaissent quatre perspectives.

Tout d'abord, nous avons pu observer l'impact qu'ont les deux processus de normalisation (l'unification et la projection) en terme d'ajout/suppression d'éléments sur les jeux de données, mais aussi de création de groupes d'éléments à égalité, ou encore de contre-performance des algorithmes positionnels sur les données unifiées. Une perspective intéressante serait de décrire ces deux processus au sein d'un même formalisme et, en étendant les travaux de [BBN13] aux classements avec égalités, de pouvoir choisir quels éléments doivent être conservés après la normalisation, et quels éléments doivent être retirés. Unifier ces deux processus pourrait à terme combiner leurs avantages, à savoir la conservation de l'ensemble des éléments pertinents, et la suppression des éléments non-pertinents.

Nous avions remarqué dans l'état de l'art que le pré-traitement de [BBN13] permettant de découper une instance du problème de l'agrégation de classements en plusieurs instances indépendantes avait un fonctionnement très proche de KwikSort. Nous avions remarqué que KwikSort choisissait un élément au hasard pour découper l'instance alors que le pré-traitement ne choisissait que des éléments "propres" (*cf.* Section 2.2.2.3 p.32). Cette importance dans le choix du pivot a été illustrée dans nos expérimentations où l'on a pu observer que relancer plusieurs fois l'algorithme KwikSort sur une instance pouvait grandement améliorer la qualité des résultats. L'importance du choix du pivot a mené Bo Yang, un des co-auteur de notre article [Bra+15], à étudier plus en détails les façons de choisir le pivot dans ces travaux de thèse [YAN14].

Ensuite, l'algorithme BioConsert est apparu au cours des expérimentations comme étant adapté à une très large majorité des cas d'utilisations, et cela lorsque l'on considère la distance de Kendall-τ généralisée. Son algorithme, tel que décrit dans le chapitre 2, est aisément adaptable à toute autre mesure. Même si l'on peut supposer qu'il sera tout aussi efficace avec la distance de Kendall-τ généralisée induite, une étude à grande échelle est nécessaire. Des travaux préliminaires ont cependant confirmé cette supposition et ont mené à son utilisation dans le cadre de [Sta+14], article qui sera présenté au chapitre suivant.

Finalement, et toujours en considérant BioConsert, le fonctionnement de cet algorithme consiste en une amélioration de plusieurs solutions de départ qui sont les classements du jeux de données. Une variante utilisant en lieu et place de ces classements un nombre réduit de consensus de qualité acceptable pourrait permettre de diminuer le temps de calcul, et ce en ayant un impact très faible sur la qualité du consensus produit par BioConsert. Encore une fois, des travaux préliminaires ont été menés et ont corroboré cette supposition. Il en a résulté une variante de BioConsert améliorant les résultats obtenus par plusieurs algorithmes positionnels qui a été utilisé dans le cadre de ConQuR-Bio [Bra+14], approche qui sera elle aussi présentée dans le chapitre suivant.

Résumé du chapitre 4

Dans ce chapitre, nous avons évalué plusieurs algorithmes que nous avons introduits ou adaptés pour agréger des classements en prenant en compte les égalités entre éléments. De ces évaluations, nous avons pu extraire un ensemble de recommandations permettant aux utilisateurs de choisir quels algorithmes utiliser en fonction de leurs besoins et de la nature des jeux de données. Plus précisément, nous avons généré des jeux de données de natures différentes (uniforme, avec similarité, unifié) pour évaluer les algorithmes dans des conditions contrôlées reproduisant des situations réelles. Nous avons, en particulier, modélisé une génération de données par chaîne de Markov permettant d'obtenir un spectre continue de jeux de données depuis une similarité très forte jusqu'à une similarité comparable aux jeux de données uniformément générés. Les analyses effectuées ont été, tout au long des expérimentations, confrontées aux résultats obtenus sur des jeux de données réels.

Au cours des analyses, nous avons pu observer la stabilité et la qualité des consensus calculés par BioConsert. Nous avons pu observer la rapidité à laquelle les algorithmes positionnels BordaCount, CopelandMethod peuvent produire des consensus d'une qualité acceptable, ainsi que MEDRank dans une moindre mesure. Nous avons aussi pu observer l'impact très négatif que peut avoir le processus d'unification et/ou la présence de grands groupes d'éléments à égalité sur la qualité des résultats de BordaCount et CopelandMethod. Les analyses nous ont, de plus, permis d'observer que KwikSort offrait un bon compromis entre la qualité des résultats de BioConsert et le temps de réponse très rapide des algorithmes positionnels.

Nous avons finalement présenté Rank'n'ties, une plateforme permettant de reproduire les analyses effectuées dans ce chapitre, mais aussi de les étendre via de nouveaux algorithmes et jeux de données. Cette plateforme est accessible sur `http://rank-aggregation-with-ties.lri.fr`.

Chap. 5

Applications aux données biologiques du web

Sommaire

5.1 Introduction

Les chapitres précédents ont dressé un panorama complet des algorithmes capables de générer des consensus de classements, mais aussi des distances et processus de normalisation utilisés lors de la génération de consensus. Nos contributions incluent un *guide à l'utilisateur* permettant de choisir les distances, mesures et processus de normalisation adaptés à un besoin, l'introduction d'une classification des algorithmes pour calculer des consensus et une évaluation approfondie de leurs

performances en fonction de caractéristiques clés des jeux de données à agréger.

L'objectif du chapitre présent est d'introduire nos contributions relatives à l'utilisation concrète d'algorithmes d'agrégation de classements à un contexte dans lequel la génération d'un consensus est particulièrement importante : les données biologiques issues du Web.

Contrairement aux données plus classiques du Web, les données biologiques reflètent des expertises et évoluent particulièrement rapidement. En conséquence, le classement des données biologiques disponibles sur le Web est en grande partie un problème ouvert [CBL11]. De très nombreux critères de classements peuvent en effet être considérés : fournir les plus récentes, celles issues des sources les plus fiables, celles les plus "pointées" par des liens de références croisées... Choisir un critère ou définir une combinaison de critères est une tâche difficile car la perception du classement peut différer d'un expert à l'autre.

L'agrégation de classements permet dans ce contexte de tirer au mieux partie de différentes façons alternatives de classer les données en faisant émerger les données classées en tête par différents (ensembles de) critères.

En particulier, nous avons utilisé les techniques d'agrégation de données dans deux contextes : avec des données issues de l'interrogation du portail d'accès aux bases de données majeures en biologie moléculaire (NCBI Entrez) et avec des données d'expertises relatives à la similarité de paires de *worflows* scientifiques.

Ce chapitre s'organise de la façon suivante. Dans un premier temps nous présentons ConQuR-Bio, une approche basée sur la reformulation de requêtes textuelles et du calcul de consensus entre les listes de résultats retournés par le portail Entrez du NCBI. Nous y étudierons les motivations menant à proposer une telle approche, les techniques mises en place pour calculer des consensus, l'outil ConQuR-Bio qui en découle et l'évaluation des résultats proposés par ConQuR-Bio en fonction de *Gold standard* constitués en collaboration étroite avec nos collaborateurs de l'institut Curie.

Dans un second temps, nous présentons notre approche de calcul de consensus entre experts établissant des classements des *worflows* par ordre de similarité par rapport à un *workflow* de référence. Nous y présenterons la collecte des données, le choix de distance, et l'algorithme qui résulte de ce choix. Nous évaluerons aussi en quoi les classements *consensus* calculés représentent effectivement des points de vue consensuels entre les experts.

5.2 ConQuR-Bio : consensus de reformulations

Les contributions présentées dans cette section ont été obtenues en collaboration avec Bastien Rance (Hôpital Européen Georges Pompidou). L'article [Bra+14] résultant de cette collaboration a été publié à la DILS 2014 : Conférence internationale sur l'intégration de données pour les sciences du vivant.

5.2.1 Motivation

Le nombre de bases de données biologiques publiques et la masse d'information contenue dans ces bases de données est en constante augmentation. Interroger ces bases de données est un besoin crucial si l'on veut connaître l'existant et contribuer à de nouvelles connaissances. De grandes quantités de données peuvent être facilement obtenues au travers de portails tels que Entrez de NCBI [1] [Mag+11]. Ce portail est utilisé quotidiennement par des médecins, biologistes et bioinformaticiens qui lui soumettent des requêtes faites de mots-clés (que l'on appelle ici *phrase-clé*).

Bien interroger de tel portails n'est pas une tache aisée. Deux requêtes très similaires peuvent fournir des ensembles de réponses différents, obligeant les utilisateurs à essayer différentes reformulations de leurs requêtes basées sur les synonymes, les variantes orthographiques, les différents niveaux de précision dans les termes utilisés, l'utilisation ou non de termes standardisés (au sein de terminologies tel que MeSH [Lip00] ou SNOMED CT [Ste+01]). Les résultats obtenus doivent ensuite être fusionnés, comparés, et les redondances éliminées ... Chaque réponse du portail est un classement, c'est-à-dire un ensemble de résultats classés. Dans le cas du portail Entrez le critère de classement est la pertinence qui correspond au *nombre d'occurrences de la phrase-clé dans chaque résultat.* Cependant, lorsqu'il est possible de considérer plusieurs reformulations, le nombre de données obtenues peut sensiblement augmenter et il est nécessaire de définir une stratégie pour interclasser les éléments obtenus par les différentes reformulations. La stratégie que nous suivons est fondée sur les besoins exprimés par nos collaborateurs biologistes : le classement de l'ensemble des données doit permettre de bien classer les données obtenues par plusieurs reformulations (et bien classées par celles-ci) tout en mettant en retrait les éléments (bien) classés par seulement quelques reformulations.

Il est alors primordial de proposer à la communauté une approche permettant à la fois de reformuler les requêtes en exploitant les différentes terminologies disponibles, mais aussi de proposer un classement de gènes qui est un consensus

1. http ://www.ncbi.nlm.nih.gov/entrez

des différents résultats obtenus pour chaque reformulation, le tout dans un outil travaillant à-la-volé.

En nous basant sur des cas d'utilisation recensés auprès de collaborateurs de l'Institut Curie et du *Children's Hospital of Philadelphia (PA, USA)*, mais aussi sur nos connaissances sur l'agrégation de classements, nous présentons ConQuR-Bio. Cette approche permet aux utilisateurs d'interroger *à-la-volé* les bases de données publiques du NCBI, tout en générant automatiquement toutes les reformulations possibles pour finalement fournir des gènes classés en utilisant des techniques d'agrégation de classements, c'est-à-dire où le classement tend à minimiser les désaccords avec l'ensemble des classements pris en entrée.

Le reste de cette section est organisée comme suit. Après la description en Section 5.2.2 d'un ensemble de cas d'utilisation qui ont guidé la conception de ConQuR-Bio, la Section 5.2.3 présente l'architecture de notre système, incluant les stratégies suivies pour reformuler les requêtes. Nous présentons en Section 5.2.4 la stratégie d'agrégation de classements que nous suivons. La Section 5.2.5 présente l'interface et les principales fonctionnalités du système que nous avons mis en place sur la base de l'approche ConQuR-Bio (disponibles pour une utilisation à la communauté à : `http://conqur-bio.lri.fr`). La Section 5.2.6 fournit les résultats obtenus par ConQuR-Bio sur plusieurs requêtes biologiques tandis que la Section 5.2.7 conclut.

5.2.2 Cas d'utilisation

Les cas d'utilisation que nous présentons ici sont fondés sur la façon dont nos collaborateurs de *l'Institut Curie (France)* et du *Children's Hospital of Philadelphia (PA, USA)* recherchent des informations en lien avec leurs domaines d'expertise respectifs. L'outil qu'ils utilisent est le très populaire portail Entrez [Say+11] du NCBI *Centre national des USA pour les informations biotechnologiques* (*National Center for Biotechnology Information*). Nos collaborateurs s'en servent pour rechercher des gènes associés à des maladies données associées ou non à d'autres termes raffinant la recherche. Cette recherche est faite en consultant la base de données EntrezGene [Mag+11], en se focalisant plus particulièrement sur les gènes humains. L'ensemble des recherches et résultats sont en anglais. Nous décrivons par la suite quatre utilisations types que nous avons recensées.

5.2.2.1 Cas d'utilisation 1 : Les reformulations équivalentes

Considérons le cas où un utilisateur recherche des informations sur le cancer de l'utérus. Pour ce faire il va entrer *cervix cancer* dans le champ de saisie du portail EntrezGene. Le résultat est alors une liste de 460 gènes. La recherche d'information aurait pu être formulée différemment, en utilisant le mot-clef *cervical cancer* ou

encore *cancer of the cervix*, qui sont deux synonymes officiels pour dénoter le cancer de l'utérus. Chacune de ces recherches aurait alors retourné un moins grand nombre de résultats (*cf.* Table 5.1) mais susceptibles de fournir de nouveaux résultats.

Terme	Nombre de résultats
cervix cancer	460
cervical cancer	20
cancer of the cervix	2

TABLE 5.1 – Nombre de gènes retournés par le portail d'EntrezGene en utilisant différents synonymes officiels du cancer de l'utérus.

5.2.2.2 Cas d'utilisation 2 : Les abréviations

Un second cas d'utilisation est lié à l'utilisation d'abréviations *officielles* pour les noms de maladies. Considérons ici le *trouble du déficit de l'attention avec hyperactivité*, lorsque l'on recherche des gènes avec sa traduction anglophone *Attention deficit hyperactivity disorders*, 144 gènes sont retournés. L'abréviation anglophone officielle de cette maladie est ADHD, et utiliser cette abréviation dans une recherche retourne 109 gènes. 74 gènes sont communs aux deux ensembles de données obtenues.

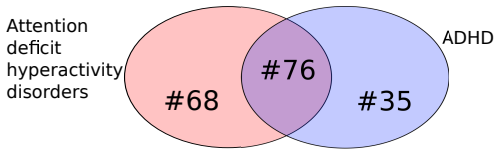

FIGURE 5.1 – Nombre de gènes retournés avec le nom de la maladie au complet, ou avec son abréviation officielle.

5.2.2.3 Cas d'utilisation 3 : Les variations linguistiques

Considérons deux utilisateurs, l'un des États-Unis et l'autre de Grande-Bretagne, ces deux utilisateurs recherchent des gènes suppresseurs de tumeurs associés au cancer du sein. Alors que l'utilisateur s'exprimant en un anglais "américain" écrit dans le champ de recherche *breast cancer tumor suppressor*, l'utilisateur britannique écrit *breast cancer tumo**u**r suppressor*. Cette variation orthographique provoque de grosses différences lorsque l'on recherche dans la base de données EntrezGene : 681 gènes sont retournés en utilisant *tumor* alors que 291 gènes le sont avec *tumo**u**r*, et 246 gènes sont communs au deux requêtes.

5.2.2.4 Cas d'utilisation 4 : Les formulations plus précises

Dans ce dernier cas d'utilisation nous considérons les maladies présentant une variété de sous-types, ces derniers étant habituellement liés à différents phénotypes ou niveaux d'expression de phénotypes associés à la maladie. Par exemple, dans le cas du cancer colorectal lorsque c'est une forme héréditaire et sans polypose, on le connaît alors comme étant le *Syndrome de Lynch*. Il est intéressant de noter qu'interroger EntrezGene avec *Hereditary Nonpolyposis Colon Cancer* et *Lynch syndrome* permet de retourner respectivement 1 et 6 gènes qui n'ont pas été retournés en recherchant avec le mot-clé *colorectal cancer*.

5.2.2.5 Esquisse du besoin

Les quatre cas d'utilisation présentés font apparaître clairement le besoin de reformulation automatique et transparente pour l'utilisateur. Face au nombre de réponses obtenues pour chaque requête, nombre d'autant plus qu'un grand qu'il y a de reformulations pouvant être proposées, les utilisateurs doivent être guidés dans leur façon de considérer les gènes retournés par les différentes reformulations, et l'ordre dans lequel les gènes doivent être considérés. L'originalité de notre approche tient au fait que l'on considère des reformulations alternatives pour la requête de l'utilisateur, et que l'on exploite les résultats obtenus pour chacune de ces reformulations afin de calculer un classement qui est un consensus de chacun des résultats. En résumé, les gènes obtenus par un grand nombre de reformulations doivent être mis en avant par rapport à ceux retournés par peu de reformulations, d'autant plus s'ils sont placés en début de classement.

5.2.3 L'approche ConQuR-Bio

Nous présentons ici ConQuR-Bio, pour *Consensus de classements avec reformulation de requêtes pour les données biologiques.* (*Consensus ranking with Query Reformulation for Biological data*), qui vise à aider les utilisateurs dans la recherche de gènes associés à une maladie donnée en considérant différentes reformulations de chaque requête et en exploitant ces reformulations pour ordonner la liste des résultats. Plus précisément, notre approche s'inscrit dans le cadre de l'expansion de requêtes (*query expansion*) en reformulant la requête de l'utilisateur pour enrichir les résultats. Notre approche s'inscrit aussi dans le cadre de l'agrégation de classements car elle prend en entrée plusieurs classements (plusieurs listes de gènes, chacune fournie par une reformulation) et produit un consensus (un classement) contenant tous les gènes pour lequel les désaccords avec les classements en entrées sont minimisés.

Nous présentons ci-après l'architecture principale de ConQuR-Bio, puis nous détaillons deux modules importants de notre approche : le *module de reformulation* et le *module de génération requêtes*.

5.2.3.1 Architecture générale

L'utilisation classique de ConQuR-Bio considère un utilisateur fournissant une phrase-clé k (c'est-à-dire une liste de mot-clés). Cette phrase-clé est alors envoyée (flèche 1 dans la Figure 5.2 p.143) au *Module de Reformulation* qui décompose k en une liste T de termes et utilise différentes terminologies pour générer pour chaque terme un ensemble S de synonymes (*cf.* Section 5.2.3.2). S est ensuite transmis (via la flèche 2) au *Générateur de Requêtes* pour former un ensemble Q de requêtes (*cf.* Section 5.2.3.3). Les requêtes de Q sont ensuite soumises (flèche 3) au moteur de recherche sélectionné, dans notre cas le portail du NCBI sur la base de données EntrezGene afin d'avoir des listes de gènes triées par "pertinence". Une fois l'ensemble R des listes de résultats obtenus en réponse aux requêtes Q, elles sont envoyées 4 au *Module agrégation de classements* (*Median Ranking Module*) qui est chargé de calculer un unique classement consensuel, et donc un ordonnancement des réponses (*cf.* Section 5.2.4). Finalement, 5 le module de *Formatage des Résultats* enrichit les classements de gènes avec leurs noms et descriptions afin de fournir à l'utilisateur une expérience semblable à celle proposée sur le moteur de recherche de référence, le portail du NCBI pour la base EntrezGene.

Des paramètres permettent à l'utilisateur d'adapter ConQuR-Bio à ses besoins, il peut par exemple sélectionner une espèce différente (par défaut on recherche des informations relatives aux gènes humains). Une option permet de faire une recherche plus approfondie (*search deeper*) permettant de considérer un nombre plus important de reformulations (*cf.* Section 5.2.3.2).

5.2.3.2 Le Module de Reformulation

Un des deux modules principaux de ConQuR-Bio est le *Module de reformulation*. Ce module prend en entrée une phrase-clé, la découpe en une liste de termes, et retourne des ensembles de reformulations pour chaque terme. Ce module utilise plusieurs terminologies biomédicales intégrées dans une base de données commune et mise à disposition par l'UMLS® [Bod04]. Les terminologies et les données qu'elles contiennent sont décrites ci-après, suivies de la présentation des processus qui les utilise. La sélection des terminologies à considérer s'est faite en collaboration étroite avec Bastien Rance (Hôpital Européen Georges Pompidou) qui a effectué un séjour postdoctoral de trois ans au NLM (*National Library of Medicine*).

Terminologies utilisées. ConQuR-Bio utilise le Metathesaurus de *Unified Medical Language System® (UMLS)*, dans sa version courante, notons que les évaluations fournies dans les sous-sections suivantes du présent mémoire ont été faites avec la version 2013AB de l'UMLS. Notre approche utilise 5 terminologies permettant de couvrir un large éventail de domaines biomédicaux :

1. MeSH [Lip00] est développé au NLM dont les termes sont utilisés notamment pour rédiger et indexer les publications de PubMed [2].
2. SNOMED CT® [Ste+01] est une terminologie clinique utilisée à une échelle mondiale et qui peut notamment être utilisée comme terminologie pour décrire les dossiers médicaux.
3. ICD 9 CM est une classification internationale des maladies (ICD) adaptée au domaine clinique (CM) ; la version 9 a été utilisée dans le système hospitalier des États-Unis jusqu'en 1999 afin de décrire les causes de mortalité.
4. ICD 10 CM est la version actuellement en vigueur pour la classification internationale des maladies.C'est aussi la version utilisée aux États-Unis pour décrire les causes de mortalité.
5. OMIM, pour Héritage mendélien chez l'Homme (*Online Mendelian Inheritance in Man*), est un catalogue de toutes les maladies génétiques connues dans le génome humain.

Au sein du Metathesaurus les concepts peuvent être des maladies, des substances, ou des symptômes. Ils ont des attributs tels que le nom ou la définition, et sont liés à d'autres concepts par des liens typés comme "a pour sous-type" (*has child relationship(...) (CH)*) ou encore "à une relation étroite" (*has a narrower relationship (RN)*). Enfin, chaque concept est catégorisé avec au moins un *type sémantique* (sur 133 types) du *Réseau Sémantique de l'UMLS* (*UMLS Semantic Network*) [Med09]. Ces concepts sont organisés de façon arborescente.

En Figure 5.3, on présente une partie des informations du Metathesaurus relative au concept C0678222. Ce concept a pour nom *carcinome du sein* (*Breast Carcinoma*). Son type sémantique est *Neoplastic Process* qui est un sous-type de "Disease or Syndrome". Le concept "carcinome du sein" (C0678222) est lié à un autre concept C1458155, Neoplasms mammaires (*Mammary Neoplasms*), par un lien "à une relation étroite" (*has a narrower relationship (RN)*).

Nous décrivons maintenant le processus que nous suivons pour exploiter les terminologies décrites. Ce processus est composé de 5 étapes : l'identification des

2. PubMed est un service de la bibliothèque de nationale de médecine des USA (US NLM) qui propose un accès à Medline, une base de données indexant les citations et résumés des publications relatives aux sciences du vivant (Source : `http://www.nlm.nih.gov/services/pubmed.html`)

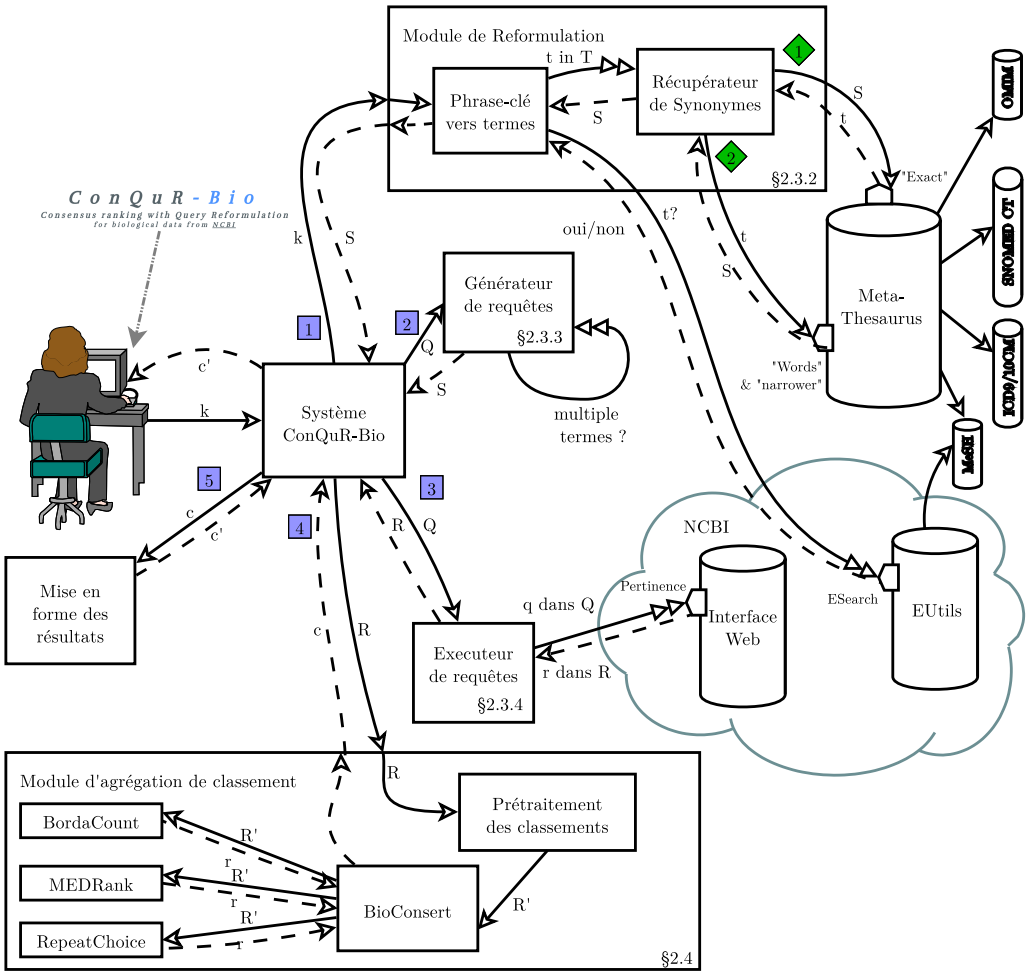

FIGURE 5.2 – Architecture de ConQuR-Bio. Les flèches pleines représentent des requêtes alors que les flèches en pointillés sont des réponses. Les flèches à deux têtes indiquent qu'il peut y avoir plusieurs appels. Lorsque plusieurs actions doivent être faites successivement, elles sont numérotées dans un carré bleu. Lorsque plusieurs actions peuvent être exécutées alternativement, elles sont numérotées dans un losange vert.

termes MeSH dans la phrase-clé, le choix du mode de reformulation, l'identification des synonymes et/ou l'identification des termes proches, et finalement un filtrage sémantique des reformulations obtenues.

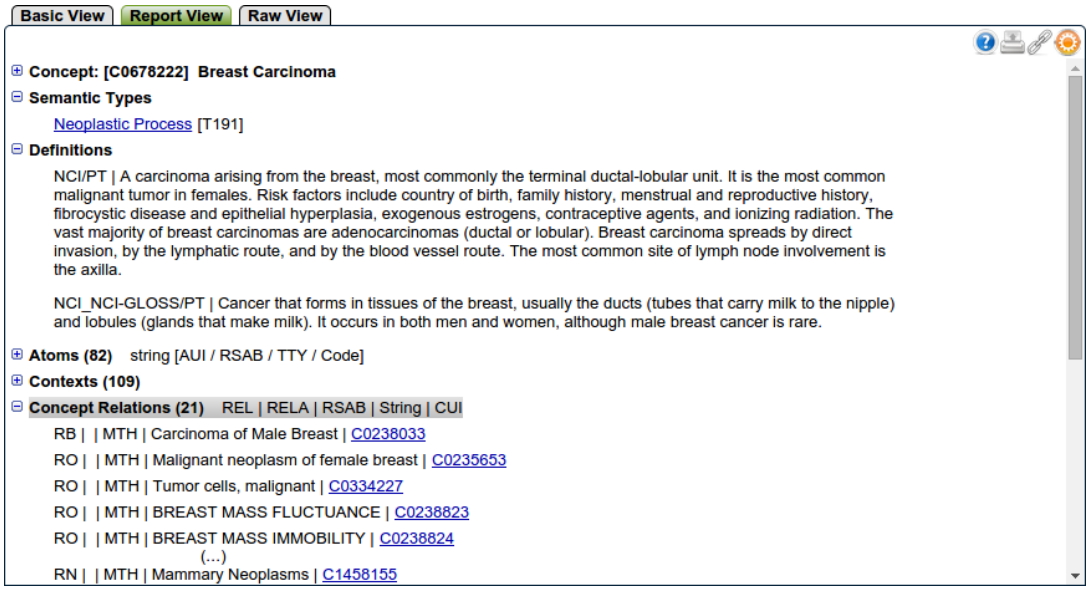

FIGURE 5.3 – Extrait des informations présente dans le Metathesaurus pour le concept C0678222, le cancer du sein. On y trouve son type sémantique, une définition mais aussi des synonymes.

Identification des termes MeSH dans la phrase-clé. L'objectif de cette étape est d'identifier les termes MeSH les plus longs possibles dans la phrase-clé. En d'autres termes, on recherche un terme MeSH tel qu'aucune concaténation de ce terme avec un des mots-clés voisins dans la phrase clé n'appartienne à MeSH. L'exemple ci-dessous fournit une illustration de ce processus

Exemple 5.1. Considérons que la phrase-clé de l'utilisateur est "breast cancer oncogene" (oncogène pour le cancer du sein). Plusieurs termes sont présents : "breast", "cancer", "breast cancer", "oncogene". La décomposition en deux termes, "breast cancer" et "oncogene" est celle qui considère les termes les plus longs.

$$\overbrace{\underbrace{breast}_{id\ D001940}\ \underbrace{cancer}_{id\ D009369}}^{id\ D001943}\ \underbrace{oncogene}_{id\ D009857}$$

Mode de reformulation Une fois les termes MeSH identifiés, la reformulation des termes peut être effectuée suivant deux modes. Dans le premier mode, ConQuR-Bio utilise l'UMLS () pour trouver les synonymes des termes MeSH identifiés dans la phrase-clé originale. C'est le mode utilisé par défaut. Dans le

second mode, ConQuR-Bio recherche les synonymes mais aussi des termes plus spécifiques (*narrower terms*). Ce second mode (2) est utilisé dans deux contextes : lorsqu'il n'existe qu'une unique reformulation pour un terme (ce mode permet alors de rechercher des reformulations additionnelles) ou lorsque l'utilisateur souhaite exploiter l'option de recherche approfondie (*Search Deeper*).

Identification des synonymes (flèche 1). L'identification des synonymes se fait l'utilisation de l'API proposée par l'UMLS. Par défaut, nous utilisons le mode de correspondance exacte (*exact match*) pour trouver les concepts UMLS associés à chaque terme. De ces concepts, on extrait tous les synonymes depuis les terminologies SNOMED CT, ICD9, ICD10 and MeSH. Par exemple, le terme *cervix carcinoma* (carcinome du col de l'utérus) est associé au concept UMLS C0302592. Ce concept inclut plusieurs synonymes dont *Cancer of cervix* (cancer du col de l'utérus) depuis SNOMED CT et *Uterine Cervical Cancer* (Cancer du col utérin) depuis MeSH.

Identification des termes proches (flèche 2). Un mode de recherche alternatif est disponible dans le Metathesaurus, et permet de trouver des termes proches (dans le sens d'une organisation hiérarchique des concepts), ces derniers étant plus précis que le ou les termes utilisé(s) dans la requête originale. On exploite aussi les synonymes des termes proches. Ce mode correspond à l'utilisation de la stratégie de recherche *word* dans l'UMLS API. Par exemple, en utilisant le mode "word" avec le terme "Long QT syndrome" (concept UMLS C0023976), il est possible d'identifier plusieurs concepts proches, incluant le "Long QT syndrome type 1" (concept UMLS C0035828) , concept pour lequel le "Syndrome Romano-Ward" est un synonyme.

Filtrage sémantique. Pour répondre aux cas d'utilisation considérés dans ConQuR-Bio relatifs à la recherche de gènes pour à des maladies, nous considérons les concepts du Metathesaurus issus du groupe sémantique "Troubles" (*Disorders*).

5.2.3.3 Le module "Générateur de requêtes"

Le module *Générateur de requêtes* produit des requêtes depuis les synonymes rassemblés par le module de reformulation pour chaque terme identifié dans la phrase-clé (*cf.* Section 5.2.3.2). Ces requêtes sont construites afin de simuler un utilisateur se connectant directement au portail du NCBI et interrogeant manuellement le portail pour chaque synonyme. Par exemple pour le terme "cancer", une requête sera générée pour rechercher avec "cancer", une autre avec "carcinome malin", une autre avec "tumeur maligne", ou encore "maladie néoplasique maligne".

Lorsque la phrase-clé a été décomposée en plusieurs termes, nous considérons le produit cartésien des reformulations de chaque terme (*cf.* Exemple 5.2), c'est-à-dire que l'ensemble des combinaisons de synonymes de chaque terme sera utilisé pour interroger le NCBI.

Exemple 5.2. On considère ici que l'utilisateur a fourni en entrée la phrase-clé "breast cancer oncogene". Cette phrase-clé a été découpée (*cf.* Exemple 5.1 p.144) en deux termes "breast cancer" et "oncogene", chacun ayant respectivement les ensembles de synonymes B et O. Le produit cartésien des synonymes forme alors l'ensemble R des reformulations qui seront utilisées pour interroger le portail du NCBI.

$$\underbrace{\left\{\begin{array}{l}-\text{breast cancer}\\-\text{mammary carcinoma}\\-\text{malignant tumor of breast}\end{array}\right\}}_{B\text{ : Ensemble des reformulations de "breast cancer"}} \times \underbrace{\left\{\begin{array}{l}-\text{oncogene}\\-\text{mutation}\end{array}\right\}}_{O\text{ : Reformulations pour "oncogene"}} \Rightarrow \underbrace{\left\{\begin{array}{l}-\text{breast cancer oncogene}\\-\text{breast cancer mutation}\\-\text{mammary carcinoma oncogene}\\-\text{mammary carcinoma mutation}\\-\text{malignant tumor of breast oncogene}\\-\text{malignant tumor of breast mutation}\end{array}\right\}}_{\text{R : Reformulations générées}}$$

5.2.3.4 Le module d'interrogation en ligne

Le module d'interrogation en ligne prend en entrée l'ensemble des reformulations de la phrase-clé et interroge pour chaque reformulation le portail du NCBI comme un utilisateur le ferait. Le classement des résultats est fait par pertinence (*Relevance*). Plus précisément pour une recherche avec le terme "breast cancer" pour des gènes de l'espèce humaine, la requête entrée est "breast cancer AND human[Organism]".

5.2.4 Le module d'agrégation de classements

Le module d'agrégation de classements est un des modules majeurs de ConQuR-Bio car il permet de proposer à l'utilisateur un classement unique construit à partir des différents classements obtenus pour chaque reformulation.

Les classements considérés en entrée sont des listes d'éléments (ici des gènes). Le module produit en sortie un **consensus**, c'est-à-dire un classement contenant l'ensemble des éléments présent dans chacun des classements fournis en entrée, ordonnés de façon à ce que le nombre de désaccords entre ce consensus et les classements en entrée soit minimisé.

Des classements incomplets en entrée. Comme indiqué dans les cas d'utilisation en Section 5.2.2 deux reformulations d'une phrase-clé peuvent ne pas retourner le même ensemble de données (i.e., les ensembles de gènes obtenus peuvent être différents d'une reformulation à une autre).

L'interprétation de l'absence d'éléments dans des classements est un problème récurrent lorsque l'on agrège des classements constitués de données réelles, et les stratégies adoptées varient [AM12 ; BBN13 ; CBDH11 ; Dwo+01 ; SZ09].

Choix du processus de normalisation et de la distance. En appliquant le *guide à l'utilisateur* introduit au Chapitre 3 (*cf.* Section 3.2), nous nous interrogeons sur la pertinence des gènes absents d'un classement et présents dans d'autre (choix c2 en Figure 3.1 p.53). Ces gènes étant pertinents, le guide nous dirige vers l'utilisation du processus d'unification (*cf.* Section 2.3.2 p.38). Nous nous interrogeons ensuite sur leur pertinence relative (choix c5) : doit-on considérer que l'absence de ces deux termes dans un même classement implique qu'ils sont aussi (non-)pertinents l'un que l'autre ? Dans le cadre de ConQuR-Bio, nous avons considéré que l'on ne savait pas si ces éléments avaient ou non la même non-pertinence, et qu'il fallait donc interpréter plus finement les égalités. Nous allons donc utilisé le processus d'unification, et la pseudo-distance de Kendall-τ.

La pseudo-distance de Kendall-τ. Dans le cadre de ConQuR-Bio, nous avons formalisé et utilisé une variante de la distance de Kendall-τ présentée en Section 3.2.3 (p.58) afin d'interpréter plus finement les égalités entre éléments. Le processus d'unification ajoute les éléments initialement manquants d'un classement à la fin de ce dernier dans un groupe d'*unification*. La pseudo-distance de Kendall-τ permet alors de considérer que rompre les égalités entre éléments au sein de groupes d'unification ne constitue pas un désaccord avec les données, alors que rompre une égalité présente dans les données originales est un désaccord.

Exemple 5.3. Dans les classements ci après, B et C sont absents de r_1, ils sont donc ajoutés par le processus d'unification après les autres éléments dans r'_1. La pseudo-distance de Kendall-τ permet ensuite d'exprimer que C est une fois après B mais aucune fois à égalité avec B, comme c'est le cas dans le jeu de données

original $\mathcal{R}$.

$$\underbrace{\begin{array}{l} r_1 = [\{A\}, \{D, E\}] \\ r_2 = [\{BA\}, \{E, C\}, \{D\}] \end{array}}_{\mathcal{R}} \overset{unification}{\longrightarrow} \underbrace{\begin{array}{l} r'_1 = [\{A\}, \{D, E\}, \{B, C\}_u] \\ r'_2 = [\{B\}, \{A\}, \{E, C\}, \{D\}] \end{array}}_{\mathcal{R}'}$$

Choix des algorithmes pour calculer un consensus. Suite aux expériences menées au Chapitre 4, il apparaît clairement que BioConsert est l'algorithme à utiliser dans notre contexte. Néanmoins, BioConsert peut avoir un temps de calcul trop important pour répondre aux contraintes de réponse à-la-volée que nous avons dans le contexte de ConQuR-Bio. Afin d'accélérer le temps de calcul de BioConsert, nous considérons ici seulement trois classements à utiliser comme points de départs (et non pas l'ensemble des classements fournis en entrée comme dans [CBDH11]). Ces trois classements sont les consensus calculés par trois algorithmes rapides décrits ci-après :

- BordaCount [Bor81] a été sélectionné pour sa capacité à fournir de bons résultats lorsqu'il y a peu d'égalités dans les classements d'entrées
- MEDRank [FKS03b] a été choisi car il propose des résultats de qualité correcte lorsque de larges égalités sont présentes dans les classements
- RepeatChoice [Ail10] a été sélectionne car il s'agit de l'algorithme ayant la meilleure borne d'approximation (c'est une 2-approximation) dans la catégorie des algorithmes très rapides.

Les tests effectués dans le cadre de ConQuR-Bio ont montré que cette combinaison d'algorithme rapide et de BioConsert réduisait le temps de calcul jusqu'à un facteur 100, et ce sans qu'il y ait d'impact sur la qualité des résultats.

5.2.5 Présentation de l'outil ConQuR-Bio

L'interface par laquelle on arrive sur l'outil implémentant l'approche ConQuR-Bio est présentée en Figure 5.4. Une fois la recherche lancée, une seconde interface est présentée à l'utilisateur et peut être visualisée en Figure 5.5. Elle se compose de trois zones, la zone de recherche (en haut à gauche), la zone pour suivre l'exécution de la requête est en haut à droite, finalement la partie inférieure présente les résultats.

5.2.5.1 Zone de recherche

Dans la zone de recherche (*cf.* Figure 5.4 et Figure 5.5 en haut à gauche), la phrase-clé fournie par l'utilisateur est découpée à-la-volée (*cf.* Section 5.2.3.2) et

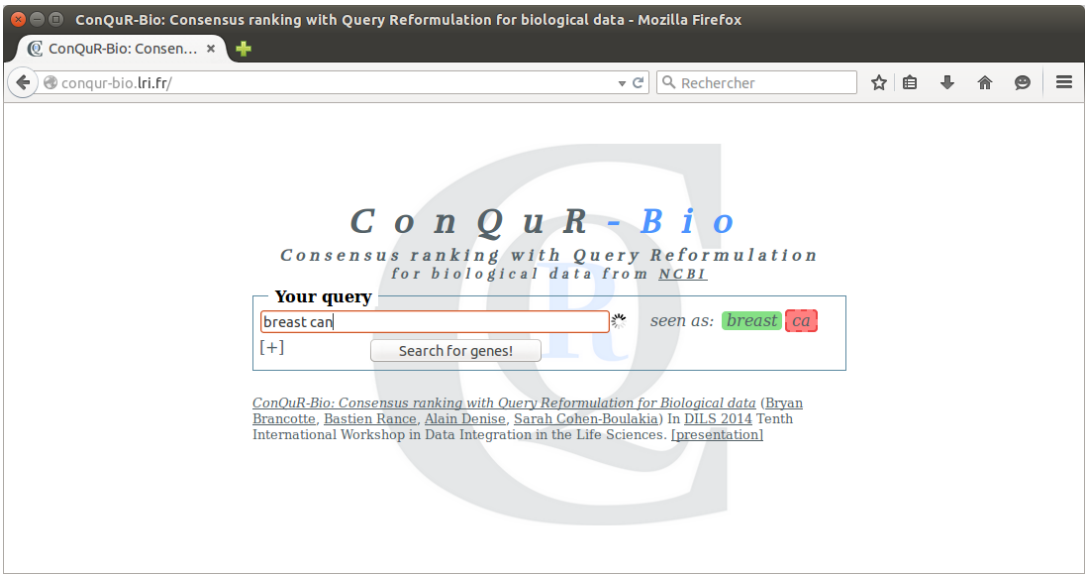

FIGURE 5.4 – Page d'accueil de ConQuR-Bio, on peut voir dans la zone de texte qu'un utilisateur est en train d'entrer "breast cancer", ConQuR-Bio recherche à la volée les termes biomédicaux existants dans ce qui est entré. Dans cette image, on remarque que la saisie textuelle n'est pas terminée et que seul "breast can" a été écrit, il en résulte que l'outil identifie *breast* mais signale que "can" n'est pas un terme biomédical connu.

est affichée dans des boites colorées a coté du champ de saisie de la phrase-clé. Un code couleur permet au système d'indiquer la façon dont sont reconnus les termes. Un terme surligné (i) en vert signifie qu'il est bien reconnu, (ii) en rouge signifie qu'il n'est pas reconnu, (iii) en orange qu'il est associé à un terme MeSH, mais avec une écriture différente. Lorsqu'un terme est identifié avec une écriture différente et est surligné en orange, il est suivit d'une coche (✓) et d'une croix permettant respectivement d'accepter la ré-écriture ou de forcer le système à utiliser le mot-clé fournit.

Des options permettent à l'utilisateur d'ajuster sa recherche (*cf.* Figure 5.5), et sont par défaut cachées (*cf.* Figure 5.4). Elles peuvent être affichées/cachées en cliquant sur "[+]"/"[-]". Ces options sont les suivantes :

- **Recherche approfondie :** L'option *"Search deeper"* permet d'utiliser plus de reformulation en utilisant les termes proches (cf. mode ? en Section 5.2.3.2).
- **L'espèce :** Cette option définit l'espèce à considerer lors de la recherche de gènes. L'espèce par défaut est *humain*.
- **Afficher les changements de rangs :**L'option *"Show rank changes"* permet d'afficher des icônes (▲, ▼, =, ☀) qui aide l'utilisateur à prendre

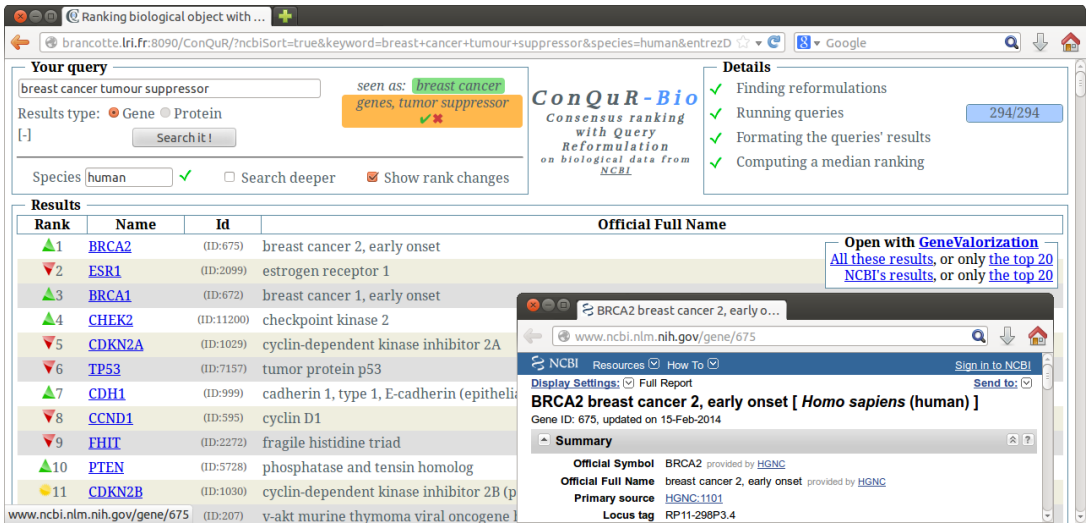

FIGURE 5.5 – L'interface de ConQuR-Bio et la fenêtre qui s'ouvre lorsque l'on clique sur le gène BRCA2.

conscience des changements dans l'ordre des résultats apportés par l'approche ConQuR-Bio, comparés à l'ordre dans lequel les résultats sont fournis par le portail Entrez du NCBI.

5.2.5.2 Zone de résultats

Nous présentons ici la zone de résultats qui est constitué de deux parties : les résultats eux-mêmes et des liens vers un autre outil, GeneValorization, que nous décrivons ci-après.

Présentation des résultats. La zone de résultats affiche un classement avec égalités de gènes. Pour chaque gène, l'affichage du nom, de l'identifiant et de la description a été fait pour reproduire l'affichage proposé par le NCBI et permettre aux utilisateurs habitués au portail EntrezGene du NCBI de naviguer dans un environnement familier. Au côté du rang, un symbole permet de visualiser l'altération du classement faite par ConQuR-Bio par rapport à une recherche classique en utilisant le NCBI. Ce symbole permet, par rapport au classement fourni par le NCBI, de savoir si le gène a été déplacé (▲) vers le haut du classement, est resté à la (=) même position, a été poussé (▼) vers le bas du classement, ou encore si ce gène (☀) a été ajouté par ConQuR-Bio. En Figure 5.5, le gène BRCA2 est classé premier par ConQuR-Bio, et était classé plus loin dans le classement du NCBI ; le gène CDKN2B n'était quant à lui pas retourné par le NCBI.

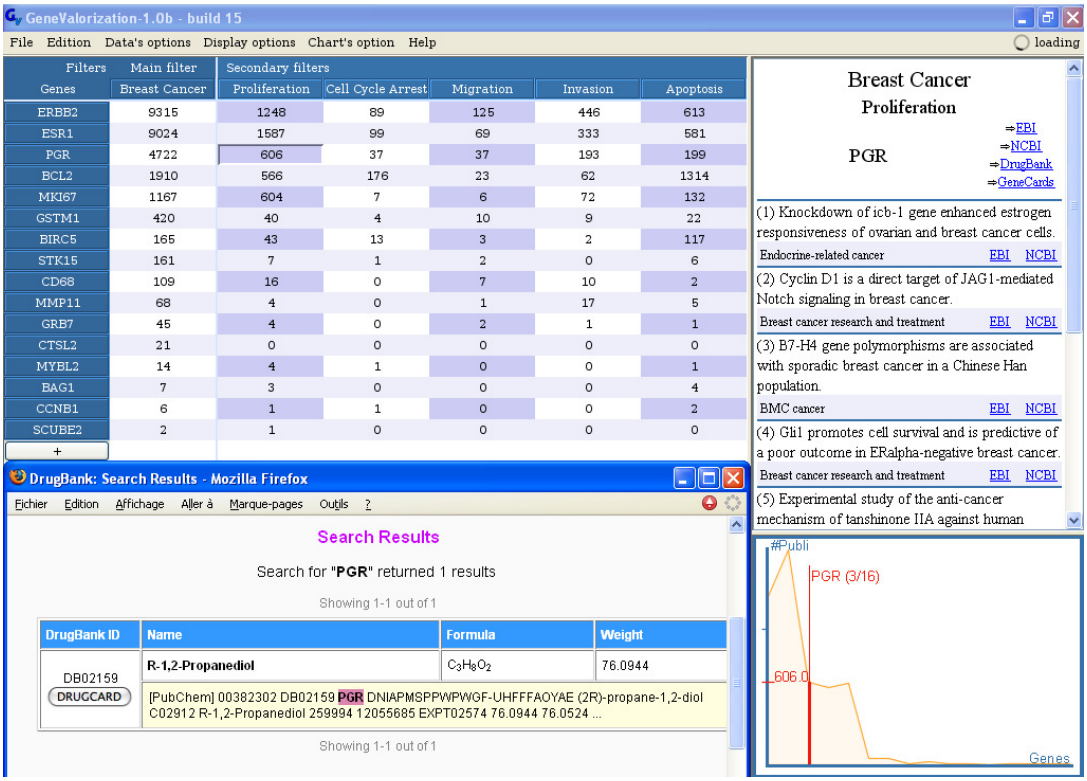

FIGURE 5.6 – Interface principale de GeneValorization. Le tableau à deux dimensions affiche dans chaque cellule le nombre de publications citant le gène de la ligne, le mot-clé de la colonne (filtre secondaire), et le mot-clé de la première colonne (filtre principale). Les publications les plus récentes retournées pour une cellule peuvent être affichées à droite en cliquant sur cette dernière. Divers liens permettent de lier les informations affichées avec des bases de données publiques, dans cet exemple, le gène PGR dans DrugBank.

Liens vers GeneValorization [Bra+11]. Une autre fonctionnalité intéressante est la possibilité pour ConQuR-Bio de fournir aux utilisateurs le nombre de publications associées à chaque gène et à la phrase-clé fournie. Cette fonctionnalité est obtenue en faisant appel à l'outil GeneValorization [Bra+11] qui permet de parcourir rapidement PubMed à la recherche de publications pour des combinaisons de gènes et de mot-clés. GeneValorization a été réalisé durant mon stage de dernière année d'école d'ingénieur en 2009, que j'ai effectué dans l'équipe Bioinformatique du LRI. J'ai conçu et développé cet outil en collaboration étroite avec des biologistes et médecins de l'institut Curie.

Le lancement de GeneValorization se fait en cliquant sur un des liens affichés dans l'encadré à droite de la zone de résultats (*cf.* Figure 5.5).

L'outil GeneValorization se présente comme une application Java qui est lancée depuis le site web `http://bioguide-project.net/gv/` [3]. GeneValorization prend en entrée une liste de gènes, un mot-clé principal, et une liste de mot-clés secondaires pour afficher dans un tableau à deux dimensions le nombre de publications citant différentes combinaisons des mot-clés et gènes. L'ensemble des recherches sont faites *à-la-volée*, c'est-à-dire que GeneValorization ne nécessite aucune base de données locale, l'ensemble des données provient de sources publiques disponibles sur le Web. Par conséquent on peut ajouter durant l'exécution de nouveaux mot-clés et gènes, les retirer, ou encore changer le mot-clé principal, l'outil rechargeant alors immédiatement les informations manquantes [4]. Dans l'interface présentée en Figure 5.6, le mot-clé principal est *Breast Cancer*, la liste de gènes contient ERBB2 et la liste de mot-clés secondaires contient *Proliferation*. Les cellules de la première colonne du tableau contiennent le nombre de publications citant à la fois le mot-clé principal et le gène de la ligne concernée. Par exemple dans la Figure 5.6 on trouve 9315 publications citant *Breast Cancer* et ERBB2. Les cellules des autres colonnes affichent le nombre de publications citant le mot-clé principale, le gène de la ligne concernée, et le mot-clé secondaire de la colonne concernée. Toujours dans la Figure 5.6 on lit que 606 publications citent *Breast Cancer*, *Proliferation* et le gène PGR.

Lorsque l'on clique sur une cellule, les titres et journaux des publications les plus récentes sont affichées à droite de l'outil. De nombreux liens permettent d'accéder aux publications et gènes dans de nombreuses bases de données publiques telle que PubMed, EntrezGene, GeneCards et DrugBank. En Figure 5.6 le lien vers DrugBank pour le gène PGR a été ouvert dans le navigateur de l'utilisateur.

GeneValorization permet de voir d'un coup d'œil l'importance qu'ont certains mot-clés et gènes par rapport à une maladie. Cet aperçu concerne l'ensemble des travaux publiés. Si maintenant on se place dans une optique de veille scientifique, GeneValorization permet aussi de ne rechercher que sur une période donnée. Ce filtrage peut être fait explicitement en spécifiant une date précise. Il peut aussi être fait de façon implicite en spécifiant de rechercher dans les x derniers mois. Dans ce second scénario, à chaque lancement de l'application, GeneValorization recherchera pour chacune des combinaisons de gènes, mot-clé principal et secondaires les articles publiés dans les x derniers mois.

3. On fait appel à la technologie *JNLP* associée au *Java Web Start* pour permettre cette intégration dans un site web.

4. Plus exactement, aussi vite que les bases de données utilisées comme PubMed ou EntrezGene le permettent, ainsi que la connexion internet de l'utilisateur.

Nom anglophone	**Traduction française**	**Type de maladie**
Attention deficit hyperactivity disorder / ADHD	Trouble du déficit de l'attention avec hyperactivité	Trouble psychiatrique
Bladder cancer	Cancer de la vessie	Cancer
Breast cancer	Cancer du sein	Cancer
Cervical/cervix cancer	Cancer du col de l'utérus	Cancer
Colorectal cancer	Cancer du colon	Cancer
Long QT Syndrome	Syndrome du QT long	Maladie cardiaque
Neuroblastoma	Neuroblastome	Cancer chez l'enfant
Prostate cancer	Cancer de la prostate	Cancer
Retinoblastoma	Rétinoblastome, cancer de la rétine	Cancer chez l'enfant
... oncogene	Oncogène (gène favorisant l'apparition de tumeurs)	
... tumor supressor	(Gène) suppresseur de tumeurs	

TABLE 5.2 – Maladies concernées par les requêtes collectées auprès d'experts (partie haute) et mot-clés certaines fois accolés aux différents cancers (partie basse).

5.2.6 Expérimentations sur des requêtes biomédicales

Nous avons testé notre approche sur un groupe de requêtes collectées auprès de collaborateur de *l'Institut Curie (France)* et du *Children's Hospital of Philadelphia (PA, USA)* en lien avec leurs domaines d'expertises respectifs. Les résultats présentés prennent en compte neuf maladies listées en Table 5.2. Pour les cancers, nous avons aussi étudié les résultats fournis lorsqu'on ajoute d'autres mots à la suite des mot-clés décrivant des cancers. Ces mots sont "gène suppresseur de tumeur" et "oncogène". Nous étudions aussi les variations orthographiques, à savoir *cervical* versus *cervix*. L'ensemble des combinaisons testées est visible dans la Figure 5.8.

Évaluer un système comme ConQuR-Bio est une tache difficile car nous devons évaluer la perception qu'a l'utilisateur des résultats. Nous avons considéré trois critères d'évaluation, en se concentrant sur les 20 premiers résultats retournés pour chaque phrase-clé. Ce nombre de résultats a été choisi par rapport au moteur de recherche du NCBI qui propose par défaut les 20 premiers gènes répondant à la requête. Le premier critère est basé sur des *Gold-standards*, c'est-à-dire dans notre cas un ensemble de gènes pertinents pour une phrase-clé, par exemple *breast cancer oncogene*. Nous avons utilisé un outil statistique classique pour évaluer si les gènes du *Gold standard* sont bien placés avant les autres, cet outil est l'aire sous la courbe de ROC [Bra97]. Les critères suivant sont d'ordre bibliométrique. Le second critère est le nombre de publications associées au vingt premiers gènes

et la phrase-clé. Le troisième est un critère de "fraîcheur des résultats" qui mesure le nombre moyen de jours qui s'est écoulé depuis la plus récente de publications associées au vingt premiers gènes. Derrière ces deux derniers critères, l'hypothèse est que des gènes très étudiés ont plus de chance d'être pertinents, et que des experts sont potentiellement plus intéressés par les informations les plus récentes et les plus à jour.

5.2.6.1 Évaluation basée sur les connaissances des experts

Dans cette sous-section, nous présentons les connaissances d'experts collectées : les *Gold standard*, puis nous présentons l'outil statistique utilisé pour mesurer la qualité des résultats, finalement nous les analysons.

Les *Gold-standards*. Pour chaque maladie, et plus généralement pour chaque phrase-clé, nous avons construit en collaboration avec des médecins la liste l_m des gènes les plus pertinents connus pour être associés à la maladie m. La "qualité" d'un consensus c_m retourné par ConQuR-Bio repose donc sur la présence des éléments de l_m dans les tous premiers résultats.

L'aire sous la courbe de ROC (AUC). Afin de comparer la qualité de la liste de gènes retournée par ConQuR-Bio par rapport à ce que retourne le moteur de recherche du portail du NCBI, et en se basant sur les nos *gold standards*, nous utilisons l'Aire sous la Courbe de ROC (ROC pour *Receiver operating characteristic* ou *caractéristique de fonctionnement du récepteur*). Cette mesure est utilisée sous l'acronyme *AUC* (*Area Under the ROC Curve*) [Bra97] dans de nombreux domaines (télécommunication, psychologie, algorithme d'apprentissage, ...), et permet de différencier l'arrivée de données attendues de celles non-attendues en fournissant en sortie une valeur sur l'intervalle $[0, 1]$, 1 étant le meilleur score.

La procédure suivie pour construire une courbe de ROC (*cf.* Figure 5.7) consiste à se placer au point (0,0), puis de parcourir successivement les données dans leur ordre d'arrivée. Pour chaque élément : s'il était recherché (vrai positif) alors on se déplace de 1 verticalement, sinon c'est un faux positif et on se déplace de 1 horizontalement. Le score (AUC) est alors l'aire sous la courbe normée par l'aire maximale. On peut aussi suivre cette procédure et ne se concentrer que sur les k premiers résultats afin d'évaluer la qualité du début de la liste des résultats.

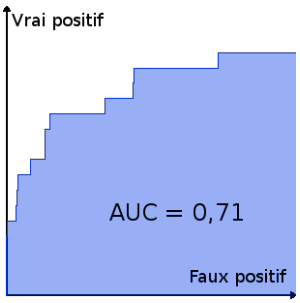

FIGURE 5.7 – Courbe de roc, et aire sous la courbe

Dans notre cas, les vrais positifs sont les données présentes dans le *Gold standard*, et les faux positifs les autres, on se concentre, comme expliqué plus haut, sur les $k = 20$

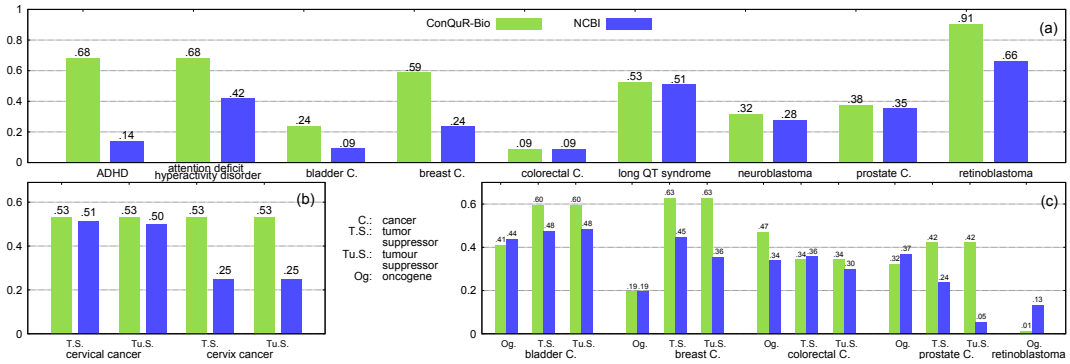

FIGURE 5.8 – L'aire sous la courbe de ROC (AUC) pour les vingt premiers gènes retournés par ConQuR-Bio et par le moteur de recherche du NCBI pour (a) les phrases-clés ne contenant qu'un terme, (b) les variations lexicales autour des gènes suppresseurs de tumeur pour le cancer de l'utérus (*cervix cancer tumor suppressor*), and (c) les autres phrases-clés. (Source : [Bra+14])

premiers résultats. Une AUC de 1 avec les k premiers résultats signifie soit que ces k résultats sont tous dans le *Gold standard*, soit que l'ensemble des éléments du *Gold standard* sont au début des résultats. Une AUC de 0 signifie qu'au mieux les éléments du *Gold standard* sont à la fin des k premiers résultats observés, ou qu'il n'y en a aucun.

Résultats. Nous présentons en Figure 5.8 les AUCs des vingt premiers résultats obtenus pour chaque phrase-clé par ConQuR-Bio et le NCBI. D'une manière générale, l'utilisation de ConQuR-Bio améliore l'AUC moyenne de 44.24%. Plus précisément, quatre points sont importants à souligner.

Premièrement, nous nous concentrons sur la Figure 5.8.a, cette dernière présente les résultats obtenus sur les phrases-clés ne contenant qu'un terme (c'est-à-dire en ne considérant que le nom de la maladie et donc sans suffixer par *oncogene* ou *tumo[u/r] suppressor*) et représente l'ensemble des *cas d'utilisation* (*cf.* Section 5.2.2). Dans cette configuration, ConQuR-Bio retourne des résultats meilleurs que le NCBI dans 88.89% des cas. Il est à noter que les résultats sont toujours au moins d'aussi bonne qualité. L'AUC moyenne est augmentée de 58.52% par ConQuR-Bio par rapport à l'AUC moyenne des résultats du NCBI.

Deuxièmement, les résultats pour phrases-clés multi-termes (Figure 5.8.b et Figure 5.8.c, cas d'utilisation 3 en Section 5.2.2.3) ont une AUC augmentée de 37.70% en moyenne en se servant de ConQuR-Bio par rapport à l'AUC des résultats du NCBI. Cette relativement moins bonne performance (37% ici contre 58% d'amélioration avec les phrases-clés mono-terme) est en réalité dû au fait que le terme *oncogene* a des reformulations qui sont moins intéressantes que d'autres. La

reformulation en "gène transformateur" (*gene transforming*) est en effet considérée comme “trop vague” par nos experts.

Troisièmement, en considérant le *trouble du déficit de l'attention* et plus particulièrement son abréviation anglophone officielle *ADHD* (cas d'utilisation 2 (*cf.* Section 5.2.2.2)), nous observons que la qualité des résultats est radicalement améliorée. L'AUC des résultats de ConQuR-Bio est de 68%, alors que l'AUC des résultats du NCBI est de seulement 14%. De plus, et comme attendu, utiliser le nom officiel ou l'abréviation officielle retourne le même résultat et a donc la même AUC, ce qui n'est pas le cas avec le NCBI. Dans le même axe où plusieurs reformulations officielles coexistent, les variations lexicales autour du cancer de l'utérus (Figure 5.8.b) montrent l'importance de prendre en compte les variations lexicales, mais aussi orthographiques : ConQuR-Bio retourne les mêmes résultats quelles que soient les variantes alors que le NCBI retourne des résultats dont l'AUC est systématiquement inférieure, mais surtout varie du simple au double.

Finalement, il y a quelques phrases-clés (*cancer du colon* (*colorectal cancer*) et *neuroblastome*) pour lesquelles seules des reformulations mettant les phrases-clés au pluriel étaient disponibles (et donc pas de vrai synonyme). Les différences entre les résultats (et AUCs) obtenus par ConQuR-Bio et par le NCBI sont donc moins importantes.

Nos évaluations ont montré que toutes les reformulations associées aux cas d'utilisation 1, 2 et 3 ont été prises en compte et que notre approche basée sur le calcul d'un consensus améliore systématiquement les réponses proposées à l'utilisateur. Cependant, nous n'avons pas encore fourni d'informations précisément pour le cas d'utilisation 4 (*cf.* Section 5.2.2.4) qui met en avant l'utilisation de termes plus spécifiques pour décrire un terme. Nous détaillons donc ci-après deux points.

Premièrement, il est intéressant de noter que les termes plus précis ont déjà été utilisés dans les résultats précédemment présentés avec le *syndrome du QT Long* car ce terme n'a pas de synonyme ou de forme plurielle. En effet, le module de reformulation tel que spécifié en Section 5.2.3.2 recherche pour un terme officiel des synonymes, et si aucun n'est trouvé alors il recherche avec des termes plus précis tels que des sous-types d'une maladie. Avec le *syndrome du QT Long*, des formes spécifiques de la maladie tel que le *Syndrome de Romano Ward* augmente l'AUC de 0.51 to 0.53.

Deuxièmement, l'utilisation des termes plus précis peut être manuellement spécifiée en utilisant l'option de *recherche approfondie* (*search deeper*) dans l'interface de ConQuR-Bio (*cf.* Section 5.2.5.1). En utilisant l'exemple illustrant le cas d'utilisation 4, dans les 11 gènes indiqués par nos experts comme étant très pertinents pour le cancer du colon (*colorectal cancer*), seuls 2 sont présents dans le top-20 du NCBI (AUC=0.09) alors que 6 gènes sont présents dans les 20 premières réponses

de ConQuR-Bio (AUC=0.43) avec la recherche approfondie.

Temps de réponse. Un dernier point à souligner est le temps nécessaire à ConQuR-Bio pour proposer des réponses. Tandis que le moteur de recherche du NCBI propose un résultat en moins de $2s$, ConQuR-Bio prend en moyenne $41s$ pour les neuf phrases-clés ne contenant qu'un terme Figure 5.8.a. Cette différence repose sur le fait que le nombre moyen de synonymes exploités par ConQuR-Bio (et donc le nombre moyen de requêtes qui doivent être exécutées) est de 17. Le temps pour fournir une réponse est, via l'interrogation du NCBI, linéaire en fonction du nombre de reformulations, et polynomial en fonction du nombre d'éléments considérés de part le calcul d'un consensus comme nous l'avons observé au Chapitre 4.

5.2.6.2 Évaluation basée sur le nombre de publications

La seconde mesure considère les 20 premiers gènes, et additionne le nombre de publications co-citant chaque gène avec la phrase-clé. Par exemple, le nombre de publications associé avec les 20 premiers gènes retournés pour le *retinoblastome* par le NCBI et ConQuR-Bio sont représentés en Figure 5.9. Cette figure montre clairement que les 20 premiers gènes proposé par ConQuR-Bio sont associés à plus de publications que ceux proposés par le NCBI.

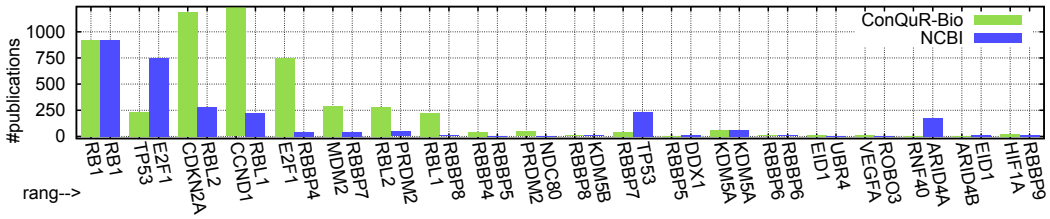

FIGURE 5.9 – Les 20 premiers gènes retournés par ConQuR-Bio et le NCBI pour la phrase-clé *retinoblastoma* ainsi que le nombre de publications associés à chacun de ces gènes. (Source : [Bra+14])

Plus généralement, sur les 28 phrases-clés étudiées, 23 sont associées par ConQuR-Bio à plus de publications que par le NCBI, et 2 phrases-clés sont associées aux mêmes nombres de publications par le NCBI et ConQuR-Bio. En moyenne le top-20 des résultats de ConQuR-Bio est associé à 56% plus de publications que ne l'est le top-20 des résultats fournis par le NCBI.

5.2.6.3 Évaluation basée sur la fraîcheur des publications associées

Alors que le nombre de publications est un facteur important pour évaluer le niveau d'attention qu'a pu recevoir un gène, un autre facteur important est

de savoir si cette attention est récente. Nous évaluons si l'attention portée à un ensemble de gènes est récente au travers des publications associés à la phrase-clé et à chaque nom de gène.

Pour une phrase-clé donnée et un ensemble de gènes, on calcule pour chaque gène le nombre de jours qui s'est écoulé depuis la plus récente des publications citant ce gène et la phrase-clé. On calcule pour chaque phrase-clé la moyenne du nombre de jours écoulé pour chacun des gènes. Cette information est un indicateur du caractère novateur (de la *fraîcheur*) de l'étude conjointe de ces gènes et d'une phrase-clé donnée.

Pour comparer la *fraîcheur* des résultats retournés par le NCBI et par ConQuR-Bio, on mesure pour chaque phrase-clé la *fraîcheur* sur les 20 premiers résultats. Sur les 28 phrases-clés étudiées, ConQuR-Bio retourne une meilleure *fraîcheur* pour 22 d'entre eux. Les résultats de ConQuR-Bio ont une *fraîcheur* 25% moins importante que ceux du NCBI, c'est-à-dire que le top-20 des gènes retourné par ConQuR-Bio est associé à un article scientifique publié depuis, en moyenne, un nombre de jours 25% plus réduit que ceux retournés par le NCBI.

5.2.7 Discussion et ouverture

Avec ConQuR-Bio nous avons fait le lien entre le domaine de *l'expansion de requêtes* (*query expansion*) et celui de l'agrégation de classements (*rank aggregation*).

Nous avons utilisé l'intégration de nombreuses terminologies au sein du metathesaurus de l'UMLS (une intégration et un système dont la pertinence a déjà été démontrée [DF+11]) pour proposer des reformulations. En utilisant deux des modes de reformulations proposés pour rechercher dans l'UMLS, nous fournissons des reformulations basées sur l'identification des termes MeSH dans la phrase-clé de l'utilisateur. Afin de proposer un consensus mettant en avant les points communs des réponses obtenues pour chaque reformulation, nous nous sommes appuyé sur la création d'une mesure étendant la généralisation de la distance de Kendall-τ. Avec cette nouvelle pseudo-métrique, nous avons adapté et combiné plusieurs algorithmes, dont les performances en qualité et en temps de calcul ont été préalablement et largement évaluées, afin de pouvoir rapidement calculer un consensus de qualité.

Nous avons comparé notre approche avec le principal portail utilisé pour interroger les données biologiques relatives aux gènes, à savoir la base de données du site EntrezGene du NCBI et sa fonction de classement basée sur la pertinence. Nous avons montré que lorsque l'on mesure la présence et l'ordre des résultats attendus (en nous basant sur des *Gold-standards*), ConQuR-Bio propose par rapport au NCBI des résultats dont l'AUC est augmentée de 44.24%. En se concentrant sur des indicateurs bibliométriques et toujours par rapport au tri par *pertinence*

du NCBI, les gènes retournés par ConQuR-Bio sont associés à 56% plus de publications, et qui ont une date de publications 25% plus récente. Enfin, nous avons implémenté l'approche ConQuR-Bio et l'avons rendu disponible au travers d'un site web à l'adresse `http://conqur-bio.lri.fr`

Nous proposons maintenant une discussion et des perspectives relatives aux différentes étapes de notre approche.

ConQuR-Bio commence par l'identification des termes MeSH dans les phrases-clés. Nous avons actuellement choisi de suivre un processus glouton (et naïf) permettant un taux de réponse très rapide, compatible avec la fonctionnalité "à-la-volée" de notre approche. Cette stratégie est tout à fait satisfaisante sur les phrases-clés évaluées. Dans les travaux à venir, nous explorerons la détection des concepts dans les phrases-clés des utilisateurs en utilisant un outil de reconnaissance de concepts comme MetaMap [Aro01] ou BioAnnotator [Sub+03]. L'utilisation de ces outils nous permettrait d'avoir des options de reformulation plus précises et adaptables aux besoins de chacun comme le niveau de granularité des reformulations ou encore les sous domaines où rechercher les reformulations. Pouvoir retourner des résultats en quelques secondes tout en augmentant leur qualité globale sera le point le plus difficile.

Le module de reformulation joue un rôle important dans la qualité des résultats. Ce module est basé sur deux éléments : l'ensemble des terminologies utilisées et la façon dont ces terminologies sont interrogées et exploitées.

Concernant les terminologies, nous utilisons actuellement des sources terminologiques intégrées dans l'UMLS, ce qui nous a permis d'avoir un volume de reformulations pertinent et réduit. Les travaux en cours comprennent la sélection d'un plus grand nombre de sources, mais aussi le fait de permettre une personnalisation des sources utilisées dans les deux principaux systèmes d'intégration de terminologies biologiques (à savoir, l'UMLS et BioPortail [Whe+11]) et ce afin de couvrir un champ plus large de domaines biologiques. Pour faire face au potentiellement grand nombre de reformulations trouvées, nous envisageons de permettre aux utilisateurs (expérimentés) de sélectionner les reformulations qui seront ou ne seront pas utilisées par notre système.

5.3 Consensus d'experts pour la similarité entre *worflows* scientifiques

Les contributions présentées dans cette section ont été obtenues dans le contexte du projet (PHC Procope) "Sharing and Optimizing Scientific Workflows" co-porté Ulf Leser (Humboldt Berlin) et Sarah Cohen-Boulakia (LRI), en collaboration avec Johannes Starlinger (Humboldt Berlin), doctorant. Ces travaux [Sta+14] ont été publiés dans les Proceedings de VLDB 2014 : Very Large Data Bases.

Nous présentons ici une seconde utilisation concrète de l'agrégation de classements, dans laquelle nous cherchons à agréger des avis d'experts. Contrairement aux cas rencontrés dans la littérature [AM12 ; Bra+14 ; Dwo+01 ; SZ09 ; BBN13] dans lesquels un élément manquant dans un classement est considéré comme moins pertinent que les éléments présents, dans notre contexte l'absence d'information est une abstention. Entre d'autres termes, un élément qui n'est pas dans le classement d'un expert n'est ni plus ni moins pertinent qu'un élément présent.

Plus précisément, les avis d'experts que nous considérons dans cette section sont relatifs à des notes données par un ensemble d'experts pour qualifier le niveau de similarité de paires de *worflows* scientifiques. Pour chaque *workflow* de référence, chaque expert classe un ensemble de *worflows* par similarité croissante relativement au *workflow* de référence. Le *modus operandi* suivi pour la collecte des évaluations de similarité permet de prendre en compte les égalités dans un classement (deux *worflows* sont aussi similaires l'un que l'autre au *workflow* de référence), mais il permet aussi que les classements ne soient pas nécessairement complets. En effet, un expert peut ne pas souhaiter s'exprimer sur l'un des *workflow*, par ce qu'il ne sait pas évaluer sa similarité par rapport au *workflow* de référence.

La section s'organise de la façon suivante : nous définissons dans un premier temps ce qu'est un *workflow*, ensuite nous présentons le besoin initial qui a mené à avoir besoin de calculer un consensus entre experts. Nous expliquons ensuite l'interprétation faite des données et le choix d'une mesure qui reflète les opinions des experts quant à la similarité de *worflows*. Dans un quatrième temps, nous détaillerons l'approche suivie par pouvoir calculer un consensus entre les classements établis par les experts. Finalement nous présentons une évaluation des consensus calculés par rapport aux classements des experts.

5.3.1 Le *workflow*, objet d'étude

Un *workflow* scientifique est défini dans [YB05] comme l'automatisation d'un processus scientifique dans lequel plusieurs taches sont organisées en fonction de leur inter-dépendance en termes de données. Les taches pouvant être réutilisées d'un *workflow* a l'autre.

Dans l'exemple ci-contre, le *workflow* permet d'obtenir à partir d'un identifiant de protéine, les processus biologiques dans lesquels elle est impliquée, ainsi que des représentations graphiques de ces processus.

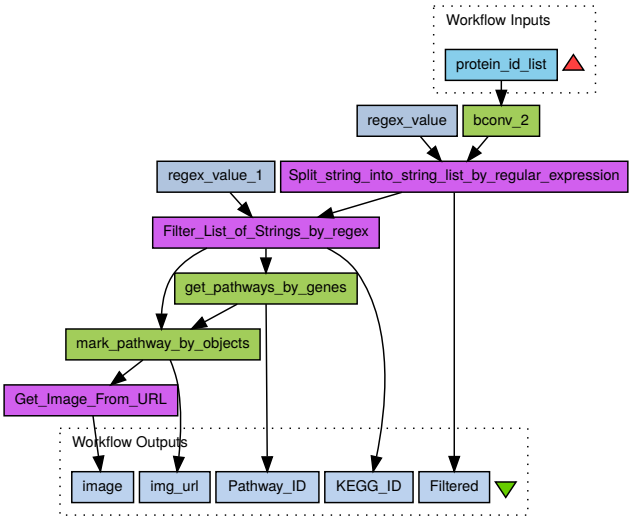

FIGURE 5.10 – En exemple de *workflow* scientifique provenant de myExperiment (ID : 1189)

Les dépôts publics tel que myExperiment [Gob+10] permettant de déposer et partager des *worflows* ont gagné en popularité. Le volume de *worflows* présents dans ces dépôts (> 2700) rend nécessaire l'utilisation de méthodes pour les comparer Cependant l'ensemble des travaux (antérieur à [Sta+14]) proposant ou évaluant des méthodes de comparaison de *worflows* ne permettent pas d'avoir un aperçu clair des méthodes à utiliser à la fois parce que ces approches sont évaluées sur des données différentes, mais aussi parce que leurs conclusions sont parfois contradictoires. Il apparaît alors le besoin de constituer des *Gold-standards* et d'évaluer chaque approche de similarité de *worflows* vis-à-vis de ces *Gold-standards*. Un *Gold standard* pour la similarité de workflows est constitué d'un *workflow* de référence et d'un ensemble de *worflows* qui sont classé par similarité vis-à-vis du *workflow* de référence.

5.3.2 Collecte des évaluations de similarités des experts

La collecte des évaluations s'est faite auprès de 15 experts provenant de 6 institutions différentes, et s'est effectuée en deux temps :

Dans un premier temps, 24 *worflows* ont été aléatoirement choisis dans les 1483 *worflows* Taverna disponibles en ligne via myExperiment et relatifs aux sciences du vivant. Pour chacun de ces *worflows* de référence, 10 *worflows* candidats ont été sélectionné aléatoirement. Chaque expert a évalué la similarité d'un ensemble de *worflows* candidats par rapport chaque workflow de référence, et ce pour 24

worflows de référence. Pour chaque *workflow* candidat l'expert évalue sa similarité avec une échelle de Likert [Lik32] à quatre niveaux : *très similaire*, *similaire*, *apparenté* (*related*) et *dissimilaire*, plus un cinquième choix *incertain* (*unsure*) afin de permettre à l'expert de ne pas étiqueter "par défaut" un *workflow* candidat en cas de doute.

Dans un deuxième temps, suite à l'exécution de plusieurs algorithmes de tri par similarité, les experts ont évalué la similarité entre chacun des worflows de référence et une liste de 21 à 68 worflows candidats. Ces workflows candidats font partie des 10 premiers workflows renvoyés au moins une fois par au moins un algorithme de similarité.

Au total, 485 paires de *worflows* ont été présentées aux 15 experts et un total de 2424 évaluations ont ainsi pu être collectées.

5.3.3 Interprétation des données et choix d'une mesure

Pour chaque *workflow* de référence chaque expert a étiqueté chaque *workflow* candidat (*cf.* Exemple 5.4). On pourrait alors produire un classement à partir des étiquettes si on avait une relation d'ordre claire entre ces dernières. Autant on peut établir que les étiquettes *très similaire*, *similaire*, *apparenté* et finalement *dissimilaire* s'ordonnent naturellement, autant pour le choix *incertain* une interprétation doit être trouvée.

Exemple 5.4.

worflows	étiquette
1017 v1 1015 v1 1018 v1 1013 v1	très similaire
1016 v1	similaire
36 v3	apparenté
307 v1	incertain
302 v1 297 v1 280 v1	dissimilaire

Étiquettes de similarité donnée au 10 *worflows* candidats par rapport au *workflow* de référence 1014 v1 et ce par l'expert 8.

Une interprétation naturelle du choix *incertain* est de dire que le *workflow* candidat doit être placé entre les *worflows apparentés* et les *worflows dissimilaires* comme c'est le cas en Exemple 5.4. Cependant, la connaissance de ce que sont les *worflows* scientifiques invalide cette interprétation. En effet, deux *worflows* peuvent avoir une (grande) partie de leurs tâches en commun et pourtant avoir des fonctions différentes. Dans ce cas un expert choisissant *incertain* ne veut pas dire que les deux *worflows* sont moins qu'*apparentés*, mais qu'il hésite entre *similaire* (pour exprimer qu'une partie du *workflow* est commune) et *dissimilaire* (pour exprimer que les deux *worflows* n'ont pas le même objectif). Pour un expert et un *workflow* de référence, on a alors un classement avec égalités (chaque groupe est fait des *worflows* candidats ayant la même étiquette) dans lequel les *worflows* étiquetés *incertain*s sont absents. Pour chaque *workflow* de référence, on a alors un jeu de données incomplet fait du classement des *worflows* candidats par chaque expert.

Exemple 5.5. Classement produit à partir des évaluations fournies par l'expert 8 par rapport au *workflow* de référence 1014 v1 détaillé en Exemple 5.4. On remarque que le *workflow* 307 v1 est absent du classement, sa position dans le consensus ne dépendant que des autres classements (non présenté ici).

$$[\{1017v1, 1015v1, 1018v1, 1013v1\}, \{1016v1\}, \{36v3\}, \{302v1, 297v1, 280v1\}]$$

On remarque que nos données et le scénario d'utilisation est celui utilisé pour illustrer le guide à l'utilisateur introduit au Chapitre 3 (*cf.* Section 3.2.1.5 p.56). En utilisant ce guide, nous remarquons qu'il faut utiliser la mesure de Kendall-τ généralisée induite (*cf.* Section 3.2.2 p.57), car elle va permettre de prendre en compte l'ensemble des opinions d'experts tout en réalisant le droit de réserve qui leur a été donné lors de la collecte des données. Cette mesure permet aussi de produire un consensus sur tous les éléments qui apparaissent au moins une fois dans un des classements du jeu de données, le tout en prenant en compte les égalités entre éléments. De façon grossière le désaccord relatif à une paire d'éléments entre deux classements est calculé classiquement si la paire apparaît dans les deux classements, et vaut zéro si un des éléments manque dans au moins un des classements.

5.3.4 Choix d'un algorithme

L'algorithme d'agrégation de classements a été choisi de façon à ce qu'il ait les caractéristiques suivantes : (1) capable de fournir des consensus de grande qualité, (2) compatible avec la mesure de Kendall-τ généralisée induite, (3) rapide (pour pouvoir être relancé dès que de nouvelles évaluations de similarité sont fournies), et (4) (sur un plan technique) ne nécessitant pas d'installer un trop grand nombre de logiciels tiers.

Suivant les recommandations du Chapitre 4, BioConsert est un excellent candidat qui répond à tous les besoins, il propose en effet des résultats de qualité (il trouve un consensus optimal dans 99.87% des jeux de données avec 10 éléments), peut être implémenté pour utiliser la Kendall-τ généralisée induite, est très efficace ($< 0.001s$ pour calculer un consensus avec 10 éléments), et son implémentation entièrement en Java le rend utilisable dans de très nombreux environnements.

5.3.5 Comparaison entre les experts et les consensus

Dans [Sta+14] nous avons comme [BG14] re-utilisé deux mesures qui permettent de comparer des classements et d'obtenir leur *exactitude* (*correctness*)

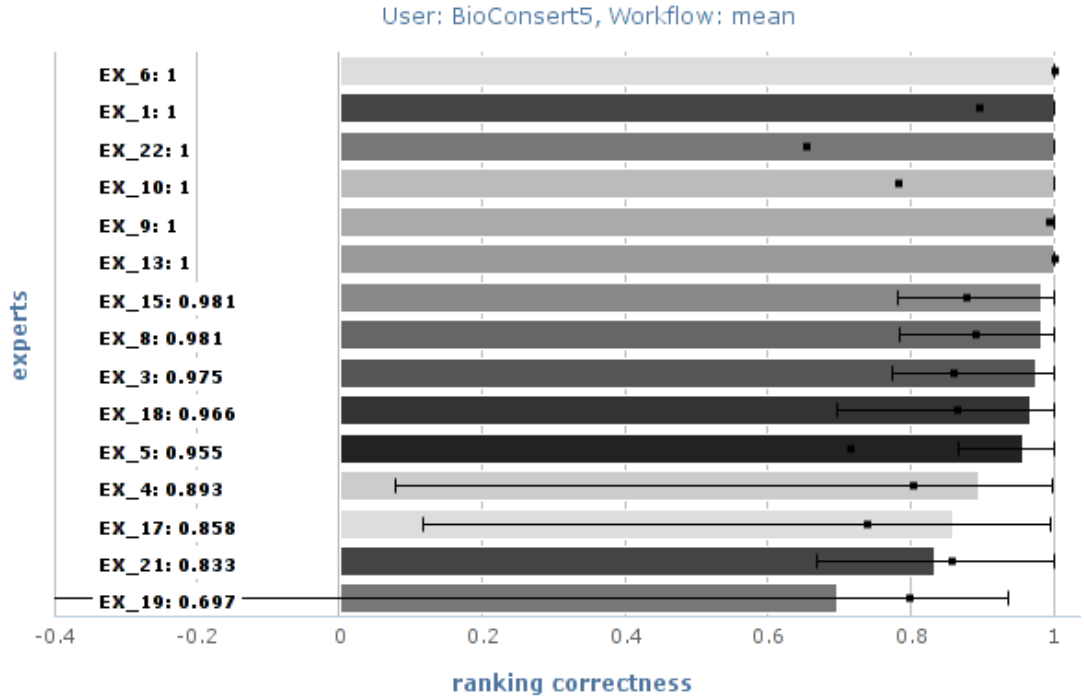

FIGURE 5.11 – Mesure d'exactitude moyenne (barre) avec les déviations standards (barre d'erreur), et la complétude moyenne (carré noir) pour chaque classement fait par un expert par rapport aux consensus calculés par BIOCONSERT.

et complétude (*complétude*) [Che+10]. Il est intéressant de noter que l'*exactitude* est équivalente au coefficient de corrélation de Kendall-*tau* dans le cas où les classements sont des permutations complètes. La complétude moyenne étant le nombre de paires de *worflows* évalué par un expert normé par le nombre de paires total.

Ces deux mesures ont été utilisées pour représenter visuellement la proximité des experts par rapport au consensus dans une image reproduite ici en Figure 5.11. En étudiant la complétude, on remarque que onze des quinze experts ont une complétude moyenne $> 80\%$. Cela signifie que les experts ont fait usage du droit de réserve, mais que dans la majorité des cas ils ont pu exprimer leur perception de la similarité entre deux *worflows*. L'exactitude des experts est très intéressante : six experts voient leurs classements dotés d'une exactitude parfaite, et onze des quinze experts voient leurs classements dotés d'une exactitude moyenne d'au moins 95.5% par rapport au consensus. Cela signifie que dans la très large majorité des cas les opinions des experts concordent (94.3% des avis), et que les consensus calculés par BIOCONSERT avec la mesure de Kendall-τ généralisée induite matérialise cette concordance d'opinions.

Résumé du chapitre 5

Dans ce chapitre nous avons présenté deux applications de l'agrégation de classements à des problématiques réelles. Nous avons montré l'intérêt de l'utilisation du guide à l'utilisateur introduit au Chapitre 3 pour identifier les mesures, distances et processus de normalisation. Nous avons aussi appliqué les recommandations établies au Chapitre 4 pour choisir les algorithmes à utiliser en fonction de caractéristiques de nos jeux de données.

Plus précisément, nous avons, dans un premier temps, présenté ConQuR-Bio [Bra+14], une approche reformulant les requêtes textuelles à l'aide de terminologies biomédicales pour la recherche de gènes. Cette approche applique l'agrégation de classements entre les résultats obtenus pour chaque reformulation, propose ensuite un consensus. Pour ce faire elle utilise le processus d'unification et la pseudo-distance de Kendall-τ afin d'interpréter finement l'absence d'élément au sein de classements, et l'égalité entre éléments dans un même classement. En se comparant au NCBI, nous avons montré que lorsque l'on mesure la présence et l'ordre des résultats attendus (en nous basant sur des *Gold-standards*), ConQuR-Bio propose meilleurs résultats avec une aire sous la courbe de ROC moyenne augmentée de 44.24%. L'approche ConQuR-Bio est rendue disponible à l'adresse `http://conqur-bio.lri.fr`.

Dans la seconde application, nous avons considéré l'agrégation de classements au consensus entre experts [Sta+14]. Dans cette application, chaque expert classe des *worflows* par similarité par rapport à un *workflow* de référence, plusieurs classements d'expert formant alors un jeu de données incomplet, et un expert ne classant pas nécessairement l'ensemble des *worflows*. Nous avons montré que les consensus obtenus par notre approche concordaient avec les avis respectifs des experts, avec une exactitude moyenne de 94.3%.

Conclusion

Durant cette thèse, notre objectif a été double. Nous avons non seulement cherché à guider l'utilisateur lors de l'utilisation des techniques de calcul de consensus entre des classements, mais aussi à appliquer ces techniques à des problématiques biologiques.

Nous avons réalisé la première partie de cet objectif en guidant l'utilisateur dans la caractérisation de son besoin de consensus, puis en le guidant lors du choix d'un algorithme pour calculer un consensus.

Nous avons ensuite appliqué à deux problématiques réelles distinctes les techniques de calcul de consensus. Nous avons, pour ce faire, proposé des outils logiciels. Nous avons proposé une approche réalisant des consensus entre listes de gènes répondant à une même requête reformulée. La seconde application a été le consensus entre experts quant à la similarité entre *worflows* scientifiques.

Caractérisation des besoins et pré-traitement des données

Tout d'abord, nous avons synthétisé nos connaissances sur les distances, mesures et processus de normalisation dans un *guide à l'utilisateur*. Ce dernier permet de caractériser le besoin de calcul de consensus d'un utilisateur et ainsi de choisir les distances, mesures et processus de normalisation à utiliser lors du calcul d'un consensus. Nous avons ainsi mis en lumière deux besoins pour lesquels aucune réponse n'avait été proposée, à savoir celui du consensus entre classements incomplets en considérant les égalités, et celui d'une interprétation fine des égalités.

Une perspective apparaît suite à l'utilisation des deux processus de normalisation, à savoir la projection et l'unification. Alors que la projection peut retirer des jeux de données des informations pertinentes, l'unification peut conserver des informations non pertinentes. Il apparaît alors le besoin de proposer un processus de normalisation combinant les avantages des deux processus en ne retirant que les informations non-pertinentes. Pour concevoir un tel processus, une piste intéressante consisterait alors à prouver que certains éléments sont nécessairement non-pertinents. Des travaux ont été menés dans [BBN13] dans le cas des classements sans égalité, et une étude dans le cas des égalités reste à mener.

Complexité du problème

Afin de pouvoir calculer des consensus dans les divers besoins mis en lumière par ce *guide à l'utilisateur*, nous avons adapté et optimisé des heuristiques existantes pour prendre en compte les égalités. Nous avons proposé un nouvel algorithme exact permettant, en temps exponentiel, de résoudre des instances de ce problème prouvé NP-difficile dans certains cas, et dont la complexité reste ouverte dans d'autres. Le problème est connu pour être NP-difficile en considérant un nombre pair d'au moins quatre classements sans égalité. Nous avons démontré que c'est aussi le cas lorsque l'on considère les égalités. Finalement, nous avons prouvé que le problème est NP-difficile aussi lorsque l'on considère des classements incomplets.

La complexité du problème de l'agrégation d'un nombre impair de classement reste un problème ouvert, qu'il y ait égalité ou non dans les classements. Cependant, nos travaux rendent valide pour l'agrégation de classements avec égalités tout résultat de complexité qui serait trouvé en ne considérant pas les égalités.

Algorithmes et recommandations

En poursuivant notre objectif de guider l'utilisateur pour le calcul d'un consensus, nous avons mené une évaluation à grande échelle des algorithmes que nous avons introduits ou adaptés. Plus précisément, nous avons généré des jeux de données de natures différentes (uniformes, avec similarité, unifiés) pour évaluer les algorithmes dans des conditions contrôlées reproduisant des situations réelles. Nous avons, en particulier, modélisé une génération de données permettant d'obtenir un spectre continu de jeux de données depuis une similarité très forte jusqu'à une similarité comparable aux jeux de données uniformément générés. Les analyses effectuées ont, tout au long des expérimentations, été confrontées aux résultats obtenus sur des jeux de données réels. De ces évaluations, nous avons pu extraire un ensemble de recommandations guidant l'utilisateur dans le choix d'un algorithme en fonction de son besoin et de la nature de ses données.

L'algorithme BioConsert est apparu comme étant celui à utiliser dans une très large majorité des cas, les améliorations apportées à cet algorithme dans ce mémoire y ont d'ailleurs contribué.

Le calcul d'un consensus pour un jeu de données de 100.000 éléments reste cependant irréalisable par ce dernier en un temps acceptable. Il convient alors de proposer de nouvelles approches. Une première piste serait de considérer une variante de BioConsert dans laquelle le déplacement d'un élément est uniquement possible dans un voisinage de sa position, cette variante pourrait radicalement réduire la complexité mémoire de BioConsert. Étendre les travaux de [BBN13]

sur le découpage des instances serait une seconde piste prometteuse. Une troisième piste serait de considérer une instance sur plusieurs niveaux de détails. Le niveau le plus fin considérant les éléments, et les niveaux supérieurs considérant des ensembles d'éléments proches dans les classements en entrée.

L'algorithme BioConsert est glouton, l'exploration de l'espace de recherche s'arrête lorsqu'un optimum est trouvé, qu'il soit global ou local. Une perspective serait alors de proposer des techniques pour sortir de ces optimums locaux, et ainsi proposer des consensus de plus grande qualité. Plusieurs stratégies existent [GP05] tel que le *recuit simulé* (*Simulated Annealing*) ou encore la *recherche à voisinage variable* (*Variable Neighborhood Search*), et il conviendrait de les appliquer à l'agrégation de classements.

Application à la recherche de gènes

ConQuR-Bio a été la première application de notre expertise sur le calcul de consensus aux données biologiques issue du Web. ConQuR-Bio considère un utilisateur recherchant des gènes *pertinents* vis-à-vis d'une requête textuelle. ConQuR-Bio exploite la richesse des terminologies biomédicales, et leur intégration au sein de l'UMLS pour reformuler cette requête et interroger *à-la-volée* le portail Entrez du NCBI pour chaque reformulation. ConQuR-Bio exploite ensuite le calcul de consensus pour rapidement retourner un classement en considérant l'absence d'éléments, et en interprétant finement les égalités entre éléments. Ce calcul rapide, de qualité, et avec une interprétation fine des égalités entre éléments, a été rendu possible par l'utilisation conjointe de BioConsert et de la pseudo-distance de Kendall-τ. ConQuR-Bio permet finalement à l'utilisateur d'évoluer dans un environement familier de recherche de données biologiques tout en bénéficiant, comme nous l'avons montré, de résultats de meilleure qualité.

La reformulation d'une requête textuelle commence actuellement par l'identification des termes MeSH au sein de cette requête. Nous suivons actuellement un processus glouton (et naïf) permettant un taux de réponse très rapide. Dans les travaux à venir, nous explorerons la détection des termes à l'aide d'outils de reconnaissance de termes comme MetaMap ou BioAnnotator. Ces outils nous permettront d'avoir des options de reformulation plus précises et adaptables aux besoins de chacun. Le point le plus difficile sera de pouvoir retourner en quelques secondes des résultats, grâce à ces outils, et ce en augmentant leur qualité globale.

Le choix des terminologies utilisées par ConQuR-Bio est actuellement fixe, il serait intéressant de laisser la possibilité à l'utilisateur de choisir ses sources. Les utilisateurs ont d'ailleurs fait remonter le besoin de choisir quelles reformulations seront utilisés par ConQuR-Bio. Nos travaux futurs intégreront l'ajout de cette fonctionnalité.

Applications au consensus entre experts

Le consensus d'experts pour la similarité entre *worflows* a été la seconde application de nos travaux sur le calcul de consensus. Dans cette approche, les experts ont évalué le niveau de similarité de paires de *worflows* scientifiques. Pour chaque *workflow* de référence, chaque expert a classé un ensemble de *worflows* par similarité croissante relativement au workflow de référence. Le *modus operandi* suivi pour la collecte des évaluations de similarité permettait de prendre en compte les égalités entre *worflows*, mais aussi le fait que les classements ne sont pas nécessairement complets. L'approche développée pour prendre en compte ce *modus operandi* a été construite autour de l'algorithme BioConsert et de la distance de Kendall-τ généralisée induite. L'évaluation des consensus calculés par rapport aux avis des experts a pu montrer que ces consensus reflétaient leurs opinions en étant en accord avec les experts dans 94.3% des cas.

Vers des outils intégrés de calcul de consensus

Afin de comparer, évaluer et comprendre les différents algorithmes pour calculer des consensus entre classements en considérant les égalités, nous avons proposé la plateforme Rank'n'ties (*cf.* `http://rank-aggregation-with-ties.lri.fr/`). Cette plateforme contient l'ensemble des données et algorithmes utilisés dans l'évaluation des algorithmes. Rank'n'ties permet d'étendre les analyses à de nouveaux algorithmes.

La plateforme Rank'n'ties permet, sans restriction, d'utiliser les algorithmes d'agrégation de classements pour calculer des consensus. L'utilisation des algorithmes est cependant circonscrite à la plateforme. Il nous semble important d'augmenter la visibilité des algorithmes permettant de calculer des consensus prenant en compte les égalités entre éléments, ainsi que les jeux de données incomplets, et de faciliter leurs utilisations. Il n'existe à ce jour aucune bibliothèque de fonctions disponible dans des dépôts publiques tel que GitHub. Un premier pas serait par exemple de rendre disponible une implémentation de BioConsert dans les dépôts du *Comprehensive R Archive Network* pour le langage *R*, un langage où les besoins semblent forts.

À moyen terme, notre but est de proposer une bibliothèque de fonction, ou un service web, intégrant à la fois notre *guide à l'utilisateur* mais aussi plusieurs algorithmes et les recommandations. Cet outil devra alors choisir entre les mesures, distances, processus de normalisation et algorithmes en fonction du besoin qui lui aura été exprimé.

Liens vers d'autres domaines

De façon plus générale, nous souhaitons étudier les liens possibles entre le calcul de consensus et des problèmes proches tels que la planification de taches dans des grilles de calculs, le retrait d'un nombre minimal d'arcs pour rendre un graphe acyclique, ou encore le consensus de classification (*Consensus clustering*). Les liens entre ces domaines pourraient peut-être permettre de répondre à des questions ouvertes quant à la complexité du problème de l'agrégation de classements dans le cas impair. Ces liens pourraient aussi permettre de proposer de nouvelles heuristiques efficaces dans ces domaines.

Bibliographie

[ACN08] Nir Ailon, Moses Charikar et Alantha Newman. "Aggregating inconsistent information : ranking and clustering". In : *Journal of the ACM (JACM)* 55.5 (2008), p. 23.

[Ail10] Nir Ailon. "Aggregation of partial rankings, p-ratings and top-m lists". In : *Algorithmica* 57.2 (2010), p. 284–300.

[AM12] Alnur Ali et Marina Meilă. "Experiments with Kemeny ranking : What works when ?" In : *Mathematical Social Sciences* 64.1 (2012), p. 28–40.

[Aro01] Alan R Aronson. "Effective mapping of biomedical text to the UMLS Metathesaurus : the MetaMap program." In : *Proceedings of the AMIA Symposium*. American Medical Informatics Association. 2001, p. 17.

[BBD09] Therese Biedl, Franz J Brandenburg et Xiaotie Deng. "On the complexity of crossings in permutations". In : *Discrete Mathematics* 309.7 (2009), p. 1813–1823.

[BBN13] Nadja Betzler, Robert Bredereck et Rolf Niedermeier. "Theoretical and empirical evaluation of data reduction for exact Kemeny Rank Aggregation". In : *Autonomous Agents and Multi-Agent Systems* (2013), p. 1–28.

[Bet+11] Nadja Betzler, Jiong Guo, Christian Komusiewicz et Rolf Niedermeier. "Average parameterization and partial kernelization for computing medians". In : *Journal of Computer and System Sciences* 77.4 (2011), p. 774–789.

[BG14] Ralph Bergmann et Yolanda Gil. "Similarity assessment and efficient retrieval of semantic workflows". In : *Information Systems* 40 (2014), p. 115–127.

[BHP04] Andrey Balmin, Vagelis Hristidis et Yannis Papakonstantinou. "Objectrank : Authority-based keyword search in databases". In : *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*. VLDB Endowment. 2004, p. 564–575.

[BK09] Jacob P Baskin et Shriram Krishnamurthi. "Preference aggregation in group recommender systems for committee decision-making". In : *Proceedings of the third ACM conference on Recommender systems*. ACM. 2009, p. 337–340.

[BM15] Bryan Brancotte et Robin Milosz. *Rank aggregation with ties is at least as difficult as rank aggregation without ties*. Internal report. Univ. Paris-Sud - France, 2015. url : `http://www.lri.fr/~brancotte/noTies.pdf`.

[Bod04] Olivier Bodenreider. "The unified medical language system (UMLS) : integrating biomedical terminology". In : *Nucleic acids research* 32.suppl 1 (2004), p. D267–D270.

[Bor81] J.C.de Borda. "Mémoire sur les élections au scrutin". In : *Histoire de l'academie royal des sciences* (1781), p. 657 –664.

[Bra+11] Bryan Brancotte, Anne Biton, Isabelle Bernard-Pierrot, François Radvanyi, Fabien Reyal et Sarah Cohen-Boulakia. "Gene List significance at-a-glance with GeneValorization". In : *Bioinformatics* 27.8 (2011), p. 1187–1189.

[Bra+14] Bryan Brancotte, Bastien Rance, Alain Denise et Sarah Cohen-Boulakia. "ConQuR-Bio : Consensus Ranking with Query Reformulation for Biological Data". In : *Data Integration in the Life Sciences*. T. 8574. LNCS. 2014, p. 128–142.

[Bra+15] Bryan Brancotte, Bo Yang, Guillaume Blin, Sarah Cohen-Boulakia, Alain Denise et Sylvie Hamel. "Rank aggregation with ties : Experiments and Analysis". In : *Proceedings of the VLDB Endowment* 8.11 (2015).

[Bra09] Bryan Brancotte. "Conception et implémentation de fonctions de tri pour les données biologiques issues du Web." co-encadré par Sarah Cohen-Boulakia et Jérome Azé. Mém.de mast. Université Paris-Sud XI, 2009.

[Bra12] Bryan Brancotte. "Vers un cadre unifiant des approches de tris pour les données biologiques". co-encadré par Sarah Cohen-Boulakia et Alain Denise. Mém.de mast. Université Paris-Sud XI, 2012.

[Bra97] Andrew P. Bradley. "The use of the area under the ROC curve in the evaluation of machine learning algorithms". In : *Pattern Recognition* 30 (1997), p. 1145–1159.

[BY06] Aaron Birkland et Golan Yona. "BIOZON : a system for unification, management and analysis of heterogeneous biological data". In : *BMC bioinformatics* 7.1 (2006), p. 70.

[CB+07] Sarah Cohen-Boulakia, Olivier Biton, Susan Davidson et Christine Froidevaux. "BioGuideSRS : querying multiple sources with a user-centric perspective". In : *Bioinformatics* 23.10 (2007), p. 1301–1303.

[CBDH11] Sarah Cohen-Boulakia, Alain Denise et Sylvie Hamel. "Using medians to generate consensus rankings for biological data". In : *Scientific and Statistical Database Management*. LNCS 6809. Springer. 2011, p. 73–90.

[CBL11] Sarah Cohen-Boulakia et Ulf Leser. "Next generation data integration for Life Sciences". In : *Proc. of the 25th Int. Conf. on Data Engineering (ICDE), IEEE*. 2011, p. 1366–1369.

[CDK06] Vincent Conitzer, Andrew Davenport et Jayant Kalagnanam. "Improved bounds for computing Kemeny rankings". In : *Proceedings of the 21st national conference on Artificial intelligence - Volume 1*. Boston, Massachusetts : AAAI Press, 2006, p. 620–626. isbn : 978-1-57735-281-5. url : `http://dl.acm.org/citation.cfm?id=1597538.1597638`.

[CFR06] Don Coppersmith, Lisa Fleischer et Atri Rudra. "Ordering by weighted number of wins gives a good ranking for weighted tournaments". In : *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*. ACM. 2006, p. 776–782.

[Che+10] Weiwei Cheng, Michaël Rademaker, Bernard De Baets et Eyke Hüllermeier. "Predicting partial orders : ranking with abstention". In : *Machine Learning and Knowledge Discovery in Databases*. Springer, 2010, p. 215–230.

[CK96] Stefan Chanas et Prezemysław Kobylański. "A new heuristic algorithm solving the linear ordering problem". In : *Computational optimization and applications* 6.2 (1996), p. 191–205.

[Cop51] Arthur H Copeland. "A reasonable social welfare function". In : *University of Michigan Seminar on Applications of Mathematics to the social sciences* (1951).

[Cri84] Douglas Edward Critchlow. *Metric methods for analyzing partially ranked data*. Technical Report. Stanford Univsersity, 1984.

[CSS99] William W Cohen, Robert E Schapire et Yoram Singer. "Learning to order things". In : *J Artif Intell Res* 10 (1999), p. 243–270.

[CW09] Tom Coleman et Anthony Wirth. "Ranking tournaments : Local search and a new algorithm". In : *Journal of Experimental Algorithmics (JEA)* 14 (2009), p. 6.

[Den02] Alain Denise. *Structures aléatoires modèles et analyse des génomes*. Université de Paris-Sud. Laboratoire de Recherche en Informatique [LRI], 2002.

[Det+09] Landon Detwiler, Wolfgang Gatterbauer, Brent Louie, Dan Suciu et Peter Tarczy-Hornoch. "Integrating and Ranking Uncertain Scientific Data". In : (2009), p. 1235–1238. url : `http://dl.acm.org/citation.cfm?id=1546683.1547468`.

[DF+11] Dina Demner-Fushman, Swapna Abhyankar, Antonio Jimeno-Yepes, Russell F Loane, Bastien Rance, François-Michel Lang, Nicholas C Ide, Emilia Apostolova et Alan R Aronson. "A Knowledge-Based Approach to Medical Records Retrieval." In : *TREC*. 2011.

[DG77] Persi Diaconis et Ronald L Graham. "Spearman's footrule as a measure of disarray". In : *Journal of the Royal Statistical Society. Series B (Methodological)* (1977), p. 262–268.

[Din+02] Chris Ding, Xiaofeng He, Parry Husbands, Hongyuan Zha et Horst D Simon. "PageRank, HITS and a unified framework for link analysis". In : *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM. 2002, p. 353–354.

[DK04] Andrew Davenport et Jayant Kalagnanam. "A computational study of the Kemeny rule for preference aggregation". In : *AAAI*. T. 4. 2004, p. 697–702.

[Dwo+01] Cynthia Dwork, Ravi Kumar, Moni Naor et Dandapani Sivakumar. "Rank aggregation methods for the web". In : *Proceedings of the 10th international conference on World Wide Web*. ACM. 2001, p. 613–622.

[Fag+04] Ronald Fagin, Ravi Kumar, Mohammad Mahdian, D Sivakumar et Erik Vee. "Comparing and aggregating rankings with ties". In : *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM. 2004, p. 47–58.

[Fag+06] Ronald Fagin, Ravi Kumar, Mohammad Mahdian, D Sivakumar et Erik Vee. "Comparing partial rankings". In : *SIAM Journal on Discrete Mathematics* 20.3 (2006), p. 628–648.

[FKS03a] R. Fagin, R. Kumar et D. Sivakumar. "Efficient similarity search and classification via rank aggregation". In : *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. ACM. 2003, p. 301–312.

[FKS03b] Ronald Fagin, Ravi Kumar et Dandapani Sivakumar. "Efficient similarity search and classification via rank aggregation". In : *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. ACM. 2003, p. 301–312.

[FZVC94] Philippe Flajolet, Paul Zimmerman et Bernard Van Cutsem. "A calculus for the random generation of labelled combinatorial structures". In : *Theoretical Computer Science* 132.1 (1994), p. 1–35.

[Gob+10] Carole A Goble, Jiten Bhagat, Sergejs Aleksejevs, Don Cruickshank, Danius Michaelides, David Newman, Mark Borkum, Sean Bechhofer, Marco Roos, Peter Li et al. "myExperiment : a repository and social network for the sharing of bioinformatics workflows". In : *Nucleic acids research* 38.suppl 2 (2010), W677–W682.

[GP05] Michel Gendreau et Jean-Yves Potvin. "Metaheuristics in combinatorial optimization". In : *Annals of Operations Research* 140.1 (2005), p. 189–213.

[Gra94] Ronald L Graham. *Concrete mathematics :[a foundation for computer science ; dedicated to Leonhard Euler (1707-1783)]*. Pearson Education India, 1994.

[GRFS15] Michael Y Galperin, Daniel J Rigden et Xosé M Fernández-Suárez. "The 2015 Nucleic Acids Research Database Issue and Molecular Biology Database Collection". In : *Nucleic acids research* 43.D1 (2015), p. D1–D5.

[Hau27] Felix Hausdorff. *Mengenlehre*. Walter de Gruyter Berlin, 1927.

[HT04] Florent Hivert et Nicolas M. Thiéry. "MuPAD-Combinat, an open-source package for research in algebraic combinatorics". In : *Sém. Lothar. Combin.* 51 (2004), Art. B51z, 70 pp. (electronic).

[JKS14] Marie Jacob, Benny Kimelfeld et Julia Stoyanovich. "A System for Management and Analysis of Preference Data". In : *Proc. of the VLDB Endowment* 7.12 (2014).

[Kem59] John G Kemeny. "Mathematics without numbers". In : *Daedalus* 88.4 (1959), p. 577–591.

[Ken38] Maurice G Kendall. "A new measure of rank correlation". In : *Biometrika* (1938), p. 81–93.

[Kle99] Jon M Kleinberg. "Authoritative sources in a hyperlinked environment". In : *Journal of the ACM (JACM)* 46.5 (1999), p. 604–632.

[Lik32] Rensis Likert. "A technique for the measurement of attitudes." In : *Archives of psychology* (1932).

[Lip00] Carolyn E Lipscomb. "Medical subject headings (MeSH)". In : *Bulletin of the Medical Library Association* 88.3 (2000), p. 265.

[Mag+11] Donna Maglott, Jim Ostell, Kim D Pruitt et Tatiana Tatusova. "Entrez Gene : gene-centered information at NCBI". In : *Nucleic acids research* 39.sp1 (2011), p. D52–D57.

[Med09] U.S. National Library of Medicine. "Semantic Networks". In : *UMLS Reference Manual*. 2009. Chap. 5.

[Mei+12] Marina Meila, Kapil Phadnis, Arthur Patterson et Jeff A Bilmes. "Consensus ranking under the exponential model". In : *arXiv preprint arXiv :1206.5265* (2012).

[Pag+99] Lawrence Page, Sergey Brin, Rajeev Motwani et Terry Winograd. "The PageRank citation ranking : Bringing order to the web." In : (1999).

[Ras+06] Louiqa Raschid, Yao Wu, Woei-Jyh Lee, María Esther Vidal, Panayiotis Tsaparas, Padmini Srinivasan et Aditya Kumar Sehgal. "Ranking target objects of navigational queries". In : *Proceedings of the 8th annual ACM international workshop on Web information and data management*. ACM. 2006, p. 27–34.

[RD01] Matthew Richardson et Pedro Domingos. "The Intelligent surfer : Probabilistic Combination of Link and Content Information in PageRank." In : *NIPS*. 2001, p. 1441–1448.

[Say+11] Eric W Sayers, Tanya Barrett, Dennis A Benson, Evan Bolton, Stephen H Bryant, Kathi Canese, Vyacheslav Chetvernin, Deanna M Church, Michael DiCuccio, Scott Federhen et al. "Database resources of the national center for biotechnology information". In : *Nucleic acids research* 39.suppl 1 (2011), p. D38–D51.

[SIY06] Paul Shafer, Timothy Isganitis et Golan Yona. "Hubs of knowledge : using the functional link structure in Biozon to mine for biologically significant entities". In : *BMC bioinformatics* 7.1 (2006), p. 71.

[Slo+03] Neil JA Sloane et al. *The on-line encyclopedia of integer sequences*. 2003.

[SSS78] Lynn Arthur Steen, J Arthur Seebach et Lynn A Steen. *Counterexamples in topology*. Springer, 1978. URL : `http://link.springer.com/book/10.1007/978-1-4612-6290-9`.

[Sta+14] Johannes Starlinger, Bryan Brancotte, Sarah Cohen-Boulakia et Ulf Leser. "Similarity Search for Scientific Workflows". In : *Proceedings of the VLDB Endowment* 7.12 (2014).

[Ste+01] Michael Q Stearns, Colin Price, Kent A Spackman et Amy Y Wang. "SNOMED clinical terms : overview of the development process and project status." In : *Proceedings of the AMIA Symposium* (2001), p. 662.

[Ste+15] W. A. Stein et al. *Sage Mathematics Software (Version 6.5)*. The Sage Development Team. 2015. url : `http://www.sagemath.org`.

[Sub+03] L Venkata Subramaniam, Sougata Mukherjea, Pankaj Kankar, Biplav Srivastava, Vishal S Batra, Pasumarti V Kamesam et Ravi Kothari. "Information extraction from biomedical literature : methodology, evaluation and an application". In : *Proceedings of the twelfth international conference on Information and knowledge management*. ACM. 2003, p. 410–417.

[SW01] B. Shneiderman et M. Wattenberg. "Ordered treemap layouts". In : *Information Visualization, 2001. INFOVIS 2001. IEEE Symposium on*. 2001, p. 73–78.

[SZ09] Frans Schalekamp et Anke van Zuylen. "Rank Aggregation : Together We're Strong." In : *ALENEX*. SIAM. 2009, p. 38–51.

[TS07] Ying Tu et Han-Wei Shen. "Visualizing changes of hierarchical data using treemaps". In : *Visualization and Computer Graphics, IEEE Transactions on* 13.6 (2007), p. 1286–1293.

[Var+09] Ramakrishna Varadarajan, Vagelis Hristidis, Louiqa Raschid, Maria-Esther Vidal, Luis Ibáñez et Héctor Rodríguez-Drumond. "Flexible and efficient querying and ranking on hyperlinked data sources". In : *Proceedings of the 12th International Conference on Extending Database Technology : Advances in Database Technology*. ACM. 2009, p. 553–564.

[VZW09] Anke Van Zuylen et David P Williamson. "Deterministic pivoting algorithms for constrained ranking and clustering problems". In : *Mathematics of Operations Research* 34.3 (2009), p. 594–620.

[Whe+11] Patricia L Whetzel, Natalya F Noy, Nigam H Shah, Paul R Alexander, Csongor Nyulas, Tania Tudorache et Mark A Musen. "BioPortal : enhanced functionality via new Web services from the National Center for Biomedical Ontology to access and use ontologies in software applications". In : *Nucleic acids research* 39.suppl 2 (2011), W541–W545.

[YAN14] Bo YANG. "Bioinformatics analysis and consensus ranking for biological high throughput data". Thèse de doct. Université Paris-Sud et de l'université de Wuhan., 2014.

[YB05] Jia YU et Rajkumar BUYYA. "A taxonomy of scientific workflow systems for grid computing". In : *ACM Sigmod Record* 34.3 (2005), p. 44–49.

[Oct14] OCTAVE COMMUNITY. *GNU Octave 3.8.1.* 2014. URL : `www.gnu.org/software/octave/`.

Annexe A

Nouvelle implémentation de BioConsert

```
public class BioConsert2015<T> {

  protected float p;
  protected boolean handleRankingNotOverTheSameElement;
  protected KendallTauGenerlized ktg;

  /**
   *
   * @param p
   * the cost to put to element in the same bucket in the consensus
   * while they are appart in the rankings and reciprocally
   */
  public BioConsert2015(float p) {
    this.p = p;
    this.handleRankingNotOverTheSameElement = true;
    this.ktg = KendallTauGenerlizedFactory.newInstance(p);
  }

  public Collection<List<Collection<T>>> computMedianRankings(List<List<Collection<T
      >>> rankings) {
    Map<T, Integer> elements = new LinkedHashMap<T, Integer>();
    int id = 0;
    List<T> elementsList = new Vector<T>(MedianRankingTools.getDistinctElement(rankings)
        );
    Collections.sort(elementsList, new Comparator<T>() {

      @Override
      public int compare(T o1, T o2) {
        return o1.toString().compareTo(o2.toString());
      }
    });
    for (T element : elementsList) {
      elements.put(element, id++);
    }
    Vector<T> elementsId = new Vector<T>(elements.keySet());
```

```
    int n = elements.size();
    double[][] posL = new double[n][n];
    double[][] posE = new double[n][n];

    // the ranking translated are the starting point for the algorithm, we
    // create one start point per ranking, plus one where everyone is in the
    // same bucket.
    int[][] startingRankings = new int[rankings.size() + 1][];
    int[] rankingTranslated;
    int idR = 0;
    for (List<Collection<T>> ranking : rankings) {
      rankingTranslated = new int[n];
      for (int i = 0; i < n; i++) {
        rankingTranslated[i] = -1;
      }
      int idBucket = 0;
      for (Collection<T> b : ranking) {
        for (T e : b) {
          rankingTranslated[elements.get(e)] = idBucket;
        }
        idBucket++;
      }
      for (int i = 0; i < n; i++) {
        if (rankingTranslated[i] >= 0) {
          for (int j = i + 1; j < n; j++) {
            if (rankingTranslated[j] >= 0) {
              if (rankingTranslated[i] < rankingTranslated[j]) {
                posL[i][j] += 1;
              } else if (rankingTranslated[i] > rankingTranslated[j]) {
                posL[j][i] += 1;
              } else {
                posE[i][j] += p;
                posE[j][i] += p;
              }
            }
          }
        }
      }
      startingRankings[idR] = rankingTranslated;
      idR++;
    }
    // the ranking where everyone is in the same bucket #0
    startingRankings[idR] = new int[n];

    Collection<List<Collection<T>>> bests = new Vector<List<Collection<T>>>();

    // posL[i][j] should be read like
    // "how many times i is before j", or
    // "how many times j is after i"
    // posE[i][j] and posE[j][i] should be read like
    // "how many times i and j are in the same bucket"
    int[] best = bioconsert(posL, posE, startingRankings, elementsId, rankings).iterator().next
        ();
    //we also could have returned many best rankings.
    List<Collection<T>> translatedBack = translateBack(elementsId, n, best);
    bests.add(translatedBack);
    return bests;
  }
```

```
protected List<Collection<T>> translateBack(Vector<T> elementsId, int n, int[]
    rankingTranslated) {
  List<Collection<T>> bestRanking = new LinkedList<Collection<T>>();
  for (int idBucket = 0; idBucket < n; idBucket++) {
    LinkedList<T> bucket = new LinkedList<T>();
    for (int i = 0; i < n; i++) {
      if (rankingTranslated[i] == idBucket) {
        bucket.add(elementsId.get(i));
      }
    }
    if (bucket.size() > 0)
      bestRanking.add(bucket);
  }
  return bestRanking;
}

protected void translateRanking(Map<T, Integer> elements, int n, int[] rankingTranslated, List
    <Collection<T>> ranking) {
  int idBucket = 0;
  for (int i = 0; i < n; i++) {
    rankingTranslated[i] = -1;
  }
  for (Collection<T> b : ranking) {
    for (T e : b) {
      rankingTranslated[elements.get(e)] = idBucket;
    }
    idBucket++;
  }
}

public Collection<int[]> bioconsert(double[][] posL, double[][] posE, int[][] rankingsTranslated,
    Vector<T> elementsId,
    Collection<List<Collection<T>>> rankings) {
  int n = posL.length;
  double[][] distance = new double[rankingsTranslated.length][2];
  double[][] costTmp = new double[rankingsTranslated[0].length + 1][2];
  // computing consensus from each ranking as starting point
  int[] bucketSize = new int[rankingsTranslated[0].length + 1];
  double[] deltaDistance = new double[2];
  int bucketCount = 0;
  for (int idRanking = 0; idRanking < rankingsTranslated.length; ++idRanking) {
    int[] rankingTranslated = rankingsTranslated[idRanking];
    for (int i = 0; i < bucketSize.length; i++) {
      bucketSize[i] = 0;
    }
    if (handleRankingNotOverTheSameElement) {
      boolean hasToUnify = false;
      int omegaBucket = 0;
      for (int i = 0; i < n; i++) {
        if (rankingTranslated[i] == -1) {
          hasToUnify = true;
          rankingTranslated[i] = bucketSize.length;
        } else if (rankingTranslated[i] > omegaBucket) {
          omegaBucket = rankingTranslated[i];
        }
      }
      if (hasToUnify) {
```

```
      omegaBucket++;
      for (int i = 0; i < n; i++)
        if (rankingTranslated[i] == bucketSize.length)
          rankingTranslated[i] = omegaBucket;

    }
  }
  // initial distance
  for (int i = 0; i < n; i++) {
    bucketSize[rankingTranslated[i]]++;
    for (int j = i + 1; j < n; j++) {
      if (rankingTranslated[i] == rankingTranslated[j]) {
        // same bucket
        distance[idRanking][1] += (posL[i][j] + posL[j][i]);
      } else if (rankingTranslated[i] > rankingTranslated[j]) {
        // i at the right of j
        distance[idRanking][0] += posL[i][j];
        distance[idRanking][1] += posE[i][j];
      } else {
        // i at the left of j
        distance[idRanking][0] += posL[j][i];
        distance[idRanking][1] += posE[i][j];
      }
    }
  }
  bucketCount = bucketSize.length - 1;
  while (bucketCount > 0 && bucketSize[bucketCount] == 0) {
    bucketCount--;
  }
  boolean improvment = true;
  double cost = 0;
  while (improvment) {
    improvment = false;
    for (int k = 0; k < n; k++) {// for each element
      int to = 0;

      to = searchToChangeBucket(k, rankingTranslated, bucketCount, costTmp, posL, posE,
          deltaDistance);// O(n)
      cost = deltaDistance[0] + p * deltaDistance[1];

      if (cost < 0) {
        distance[idRanking][0] += deltaDistance[0];
        distance[idRanking][1] += deltaDistance[1];
        improvment = true;
        int from = rankingTranslated[k];
        rankingTranslated[k] = to;
        bucketSize[from]--;
        bucketSize[to]++;
        to = bucketCount + 2;
        if (bucketSize[from] == 0) {
          deleteBuckets(from, rankingTranslated, bucketSize);// O(n)
          bucketCount--;
        }
      } else {
        to = searchToAddBucket(k, rankingTranslated, bucketCount, costTmp, posL, posE,
            deltaDistance);// O(n)
        cost = deltaDistance[0] + p * deltaDistance[1];
        if (cost < 0) {
```

```
            distance[idRanking][0] += deltaDistance[0];
            distance[idRanking][1] += deltaDistance[1];
            improvment = true;

            insertBuckets(to, rankingTranslated, bucketSize);

            bucketCount++;
            int from = rankingTranslated[k];
            rankingTranslated[k] = to;
            bucketSize[from]--;
            bucketSize[to]++;
            to = bucketCount + 2;
            if (bucketSize[from] == 0) {
              deleteBuckets(from, rankingTranslated, bucketSize);
              bucketCount--;
            }
          }
        }
      }
    }
  }

  Collection<int[]> ret = new Vector<int[]>(distance.length);
  double d = 0;
  for (int i = 0; i < distance.length; i++) {
    double dTmp = distance[i][0] + p * distance[i][1];
    if (ret.size() > 0 && dTmp < d) {
      ret.clear();
    }
    if (ret.size() == 0 || dTmp == d) {
      ret.add(rankingsTranslated[i]);
      d = dTmp;
    }
  }
  return ret;
}

protected void insertBuckets(int to, int[] rankingTranslated, int[] bucketSize) {// O(n)
  for (int z = 0; z < rankingTranslated.length; z++) {
    if (rankingTranslated[z] >= to) {
      rankingTranslated[z]++;
    }
  }
  int tmp = 0;
  for (int z = to; z < bucketSize.length; z++) {
    int tmp2 = bucketSize[z];
    bucketSize[z] = tmp;
    tmp = tmp2;
  }
}

protected void deleteBuckets(int from, int[] rankingTranslated, int[] bucketSize) {// O(n)
  for (int z = 0; z < rankingTranslated.length; z++) {
    if (rankingTranslated[z] >= from) {
      rankingTranslated[z]--;
    }
  }
  for (int z = from; z < rankingTranslated.length; z++) {
```

```
    bucketSize[z] = bucketSize[z + 1];
  }
  bucketSize[rankingTranslated.length] = 0;
}

protected int searchToChangeBucket(int elt, int[] rankingTranslated, int bucketCount, double[][]
    costTmp, double[][] posL, double[][] posE, double[] deltaDistance) {
  for (int i = bucketCount; i >= 0; i--)
    costTmp[i][0] = costTmp[i][1] = 0;
  deltaDistance[0] = deltaDistance[1] = 0;
  for (int i = 0; i < rankingTranslated.length; i++) {// O(n)
    if (i != elt) {
      if (rankingTranslated[i] == rankingTranslated[elt]) {
        costTmp[rankingTranslated[i]][1] += posE[i][elt];
        costTmp[rankingTranslated[i]][1] -= posL[elt][i];
        costTmp[rankingTranslated[i]][1] -= posL[i][elt];

        deltaDistance[1] += posE[i][elt];
        deltaDistance[1] -= posL[elt][i];
        deltaDistance[1] -= posL[i][elt];

        deltaDistance[0] += posL[elt][i];
        costTmp[rankingTranslated[i]][0] += posL[i][elt];
      } else if (rankingTranslated[elt] < rankingTranslated[i]) {
        costTmp[rankingTranslated[i]][0] += -posL[i][elt];
        costTmp[rankingTranslated[i]][1] += posL[elt][i] + posL[i][elt] - posE[i][elt];

        costTmp[rankingTranslated[i] + 1][0] += posL[elt][i];
        costTmp[rankingTranslated[i] + 1][1] += -posL[elt][i] - posL[i][elt] + posE[i][elt];

      } else if (rankingTranslated[elt] > rankingTranslated[i]) {
        costTmp[rankingTranslated[i]][0] += -posL[elt][i];
        costTmp[rankingTranslated[i]][1] += posL[elt][i] + posL[i][elt] - posE[i][elt];

        if (rankingTranslated[i] > 0) {
          costTmp[rankingTranslated[i] - 1][0] += posL[i][elt];
          costTmp[rankingTranslated[i] - 1][1] += -posL[elt][i] - posL[i][elt] + posE[i][elt];
        }
      }
    }
  }

  int to;
  to = rankingTranslated[elt] + 1;
  while (to <= bucketCount) {
    costTmp[to][0] = deltaDistance[0] += costTmp[to][0];
    costTmp[to][1] = deltaDistance[1] += costTmp[to][1];
    if (deltaDistance[0] + p * deltaDistance[1] < 0)
      return to;
    to++;
  }
  to = rankingTranslated[elt];
  deltaDistance[0] = costTmp[to][0];
  deltaDistance[1] = costTmp[to][1];
  to--;
  while (to >= 0) {
    costTmp[to][0] = deltaDistance[0] += costTmp[to][0];
    costTmp[to][1] = deltaDistance[1] += costTmp[to][1];
```

```
      if (deltaDistance[0] + p * deltaDistance[1] < 0)
        return to;
      to--;
    }
    deltaDistance[0] = deltaDistance[1] = 0;
    return -1;
  }

  protected int searchToAddBucket(int elt, int[] rankingTranslated, int bucketCount, double[][]
      costTmp, double[][] posL, double[][] posE, double[] deltaDistance) {
    for (int i = bucketCount + 1; i >= 0; i--)
      costTmp[i][0] = costTmp[i][1] = 0;
    deltaDistance[0] = deltaDistance[1] = 0;
    for (int i = 0; i < rankingTranslated.length; i++) {// O(n)
      if (i != elt) {
        if (rankingTranslated[i] == rankingTranslated[elt]) {
          costTmp[rankingTranslated[i]][1] += posE[i][elt];
          costTmp[rankingTranslated[i]][1] -= posL[elt][i];
          costTmp[rankingTranslated[i]][1] -= posL[i][elt];

          costTmp[rankingTranslated[i] + 1][1] += posE[i][elt];
          costTmp[rankingTranslated[i] + 1][1] -= posL[elt][i];
          costTmp[rankingTranslated[i] + 1][1] -= posL[i][elt];

          costTmp[rankingTranslated[i] + 1][0] += posL[elt][i];
          costTmp[rankingTranslated[i]][0] += posL[i][elt];
        } else if (rankingTranslated[elt] < rankingTranslated[i]) {
          costTmp[rankingTranslated[i] + 1][0] += posL[elt][i] - posL[i][elt];
        } else if (rankingTranslated[elt] > rankingTranslated[i]) {
          costTmp[rankingTranslated[i]][0] += posL[i][elt] - posL[elt][i];
        }
      }
    }

    int to;
    to = rankingTranslated[elt] + 1;
    while (to <= bucketCount + 1) {
      costTmp[to][0] = deltaDistance[0] += costTmp[to][0];
      costTmp[to][1] = deltaDistance[1] += costTmp[to][1];
      if (deltaDistance[0] + p * deltaDistance[1] < 0)
        return to;
      to++;
    }
    to = rankingTranslated[elt];
    deltaDistance[0] = 0;
    deltaDistance[1] = 0;
    while (to >= 0) {
      costTmp[to][0] = deltaDistance[0] += costTmp[to][0];
      costTmp[to][1] = deltaDistance[1] += costTmp[to][1];
      if (deltaDistance[0] + p * deltaDistance[1] < 0)
        return to;
      to--;
    }
    return -1;
  }
}
```

Annexe B

Distance de Kendall-τ généralisée en temps log linéaire

```
public class KendallTauGenerlizedMergeSortBasedForThesis implements KendallTauGenerlized {

  private float p;
  private float q;

  public KendallTauGenerlizedMergeSortBasedForThesis() { this(1, 1); }

  /**
   * @param p
   * * * cost when a pair is in the same bucket in one ranking and not
   * * * in the same in the other ranking
   * @param q
   * * * cost when two element are not in the gold standard, and in
   * * * different bucket in the results
   */
  public KendallTauGenerlizedMergeSortBasedForThesis(float p, float q) { this.p = p; this.q = q; }

  /**
   * O(6n+6n*log(2n)+2n) = O(8n+6n*log(2n))
   */
  @Override
  public <T> Map<KendallTauType, Float> evaluateResults(List<Collection<T>> elementsR1,
      List<Collection<T>> elementsR2) {
    int dst = 0, dstP = 0, dstQ = 0;
    int n;
    Map<KendallTauType, Float> map;

    int id, idB = 1;
    HashMap<T, Integer> mapPos = new HashMap<T, Integer>(elementsR1.size());
    for (Collection<T> bucket : elementsR1) {
      for (T elt : bucket)
        mapPos.put(elt, idB);
      idB++;
    }
```

```
    n = mapPos.size();
    idB = 1;
    id = 0;
    int[][] r2 = new int[mapPos.size()][2];
    int[] ret = new int[3];
    int[][] temp = new int[r2.length][r2[0].length];
    for (Collection<T> bucket : elementsR2) {
      for (T elt : bucket) {
        r2[id][0] = idB;
        r2[id][1] = mapPos.get(elt);
        id++;
      }
      idB++;
    }

    // O(2n+2n*log(2n))
    doPQCount(r2, ret, temp, true, false);// useless if no buckets
    // O(2n+2n*log(2n))
    doMergeSortCount(r2, ret, 0, r2.length - 1, temp);
    // O(2n+2n*log(2n))
    doPQCount(r2, ret, temp, false, true);// useless if no buckets

    dst = ret[0];
    dstP = ret[1];
    dstQ = ret[2];

    map = new HashMap<KendallTauType, Float>(4);
    map.put(KendallTauType.OriginalKen38, new Float(dst));
    map.put(KendallTauType.Generalized, dst + p * (dstP + dstQ));
    map.put(KendallTauType.Asymetric, dst + p * dstP + q * dstQ);
    float disagree = dst + p * dstP + q * dstQ;
    map.put(KendallTauType.SimilarityMeasure, 1F - 4F * disagree / (n * (n - 1)));
    return map;
  }

  @Override
  public <T> Map<KendallTauType, Float> evaluateResults(List<Collection<T>> c,
      Collection<List<Collection<T>>> inputRankings) {
    int dst = 0, dstP = 0, dstQ = 0;
    float agree = 0;
    Map<KendallTauType, Float> map;

    int id = 1, idB = 1;
    HashMap<T, Integer> mapPos = new HashMap<T, Integer>(c.size());
    for (Collection<T> bucket : c) {
      for (T elt : bucket)
        mapPos.put(elt, idB);
      idB++;
    }

    int[][] r2 = new int[mapPos.size()][2];
    int[] ret = new int[3];
    int[][] temp = new int[r2.length][2];
    for (List<Collection<T>> elementsR2 : inputRankings) {
      idB = 1;
      id = 0;
      for (Collection<T> bucket : elementsR2) {
        for (T elt : bucket) {
```

```
          r2[id][0] = idB;
          r2[id][1] = mapPos.get(elt);
          id++;
        }
        idB++;
      }

      for (int j = 0; j < ret.length; j++)
        ret[j] = 0;
      doPQCount(r2, ret, temp, true, false);
      doMergeSortCount(r2, ret, 0, r2.length - 1, temp);
      doPQCount(r2, ret, temp, false, true);
      dst += ret[0];
      dstP += ret[1];
      dstQ += ret[2];
    }

    map = new HashMap<KendallTauType, Float>(4);
    map.put(KendallTauType.OriginalKen38, new Float(dst));
    map.put(KendallTauType.Generalized, dst + p * (dstP + dstQ));
    map.put(KendallTauType.Asymetric, dst + p * dstP + q * dstQ);
    float disagree = dst + p * dstP + q * dstQ;
    map.put(KendallTauType.SimilarityMeasure, 1F - 4F * disagree / (inputRankings.size() *
        mapPos.size() * (mapPos.size() - 1)));
    return map;
  }

  protected boolean isSortNeeded(int[][] r2, int start, int end) {
    for (int i = start; i < end; i++) {
      if (r2[i][1] > r2[i + 1][1])
        return true;
    }
    return false;
  }

  /**
   * Complexity : O(2n*log(2n))
   *
   * @param r2
   * @param counts
   * @param start
   * @param end
   * @param temp
   */
  protected void doMergeSortCount(int[][] r2, int[] counts, int start, int end, int[][] temp) {
    if (end <= start)
      return;
    int mid = (end - start >> 1) + start;
    int left = start;
    int right = mid + 1;
    int i = 0;
    doMergeSortCount(r2, counts, start, mid, temp);
    doMergeSortCount(r2, counts, right, end, temp);

    i = start;
    while (right <= end && left <= mid) {
      if (r2[left][0] != r2[right][0]) {
        if (r2[left][1] < r2[right][1]) {
```

```
        for (int j = 0; j < temp[i].length; j++)
          temp[i][j] = r2[left][j];
        i++;
        left++;
      } else if (r2[left][1] > r2[right][1]) {
        for (int j = 0; j < temp[i].length; j++)
          temp[i][j] = r2[right][j];

        counts[0] += mid - left + 1;
        i++;
        right++;
      } else {// if (r2[left][1] == r2[right][1]) {
        if (r2[left][0] < r2[right][0]) {
          for (int j = 0; j < temp[i].length; j++)
            temp[i][j] = r2[left][j];
          i++;
          left++;
        } else {
          for (int j = 0; j < temp[i].length; j++)
            temp[i][j] = r2[right][j];
          i++;
          right++;
        }
      }
    } else {
      if (r2[left][1] < r2[right][1]) {
        for (int j = 0; j < temp[i].length; j++)
          temp[i][j] = r2[left][j];
        i++;
        left++;
      } else if (r2[left][1] > r2[right][1]) {
        for (int j = 0; j < temp[i].length; j++)
          temp[i][j] = r2[right][j];
        i++;
        right++;
      } else {// if (r2[left][1] == r2[right][1]) {
        for (int j = 0; j < temp[i].length; j++)
          temp[i][j] = r2[left][j];
        i++;
        left++;
      }
    }
  }
  while (left <= mid) {
    for (int j = 0; j < temp[i].length; j++)
      temp[i][j] = r2[left][j];
    i++;
    left++;
  }
  while (right <= end) {
    for (int j = 0; j < temp[i].length; j++)
      temp[i][j] = r2[right][j];
    i++;
    right++;
  }
  for (i = start; i <= end; i++) {
    for (int j = 0; j < temp[i].length; j++)
      r2[i][j] = temp[i][j];
```

```
    }
  }

  /**
   * Count the number of pair of elements that are tied/separated in r2
   * compared to c<br/>
   * Complexity : O(2n+doMergeSortCount(n))
   *
   * @param r2
   * @param counts
   * * * Add of q or p in this array
   * @param temp
   * @param doPelseQ
   * * * true if if count the tying of elements, false if i count the
   * * * untying
   * @param bucketSorted
   * * * are the bucket sorted ? I.e. two elements are tied in one
   * * * ranking they are in the order where they are in the second
   * * * ranking
   */
  private void doPQCount(int[][] r2, int[] counts, int[][] temp, boolean doPelseQ, boolean bucketSorted
      ) {
    if (!bucketSorted) {
      int start = 0;
      for (int j = 1; j < temp.length; j++) {
        if (r2[j][0] != r2[j - 1][0]) {
          if (start + 1 < j) {
            doMergeSortCount(r2, counts, start, j - 1, temp);
          }
          start = j;
        }
      }
      if (start + 1 < temp.length) {
        doMergeSortCount(r2, counts, start, temp.length - 1, temp);
      }
    }
    int d;
    temp[0][1] = temp[0][0] = 0;
    for (int j = 0; j < temp.length; j++) {
      for (int i = 0; i < 2; i++) {
        if (j == 0 || r2[j][i] != r2[j - 1][i])
          temp[j][i] = 0;
        else
          temp[j][i] = 1 + temp[j - 1][i];
      }
      d = temp[j][0] - temp[j][1];
      if (doPelseQ) {
        if (d > 0)// p
          counts[1] += d;
      } else {
        if (d < 0)// q
          counts[2] -= d;
      }
    }
  }
}
```