Rank aggregation with ties: Experiments and Analysis

Bryan Brancotte [*]
brancotte@lri.fr

Bo Yang [*†]
yangbo@whu.edu.cn

Guillaume Blin [‡]
guillaume.blin@labri.fr

Sarah Cohen-Boulakia [⋆* §]
cohen@lri.fr

Alain Denise [*¶]
denise@lri.fr

Sylvie Hamel [⋆♯]
hamelsyl@iro.umontreal.ca

ABSTRACT

The problem of aggregating multiple rankings into one consensus ranking is an active research topic especially in the database community. Various studies have implemented methods for rank aggregation and may have come up with contradicting conclusions upon which algorithms work best. Comparing such results is cumbersome, as the original studies mixed different approaches and used very different evaluation datasets and metrics. Additionally, in real applications, the rankings to be aggregated may not be permutations where elements are strictly ordered, but they may have ties where some elements are placed at the same position. However, most of the studies have not considered ties.

This paper introduces the first large scale study of algorithms for rank aggregation with ties. More precisely, (i) we review rank aggregation algorithms and determine whether or not they can handle ties; (ii) we propose the first implementation to compute the exact solution of the Rank Aggregation with ties problem; (iii) we evaluate algorithms for rank aggregation with ties on a very large panel of both real and carefully generated synthetic datasets; (iv) we provide guidance on the algorithms to be favored depending on dataset features.

[*]LRI (Laboratoire de Recherche en Informatique), CNRS UMR 8623 Univ. Paris-Sud - France

[†]State Key Laboratory of Virology, College of Life Sciences, Wuhan Univ., Wuhan, China.

[‡]Univ. Bordeaux, LaBRI, CNRS UMR 5800, F-33400 Talence, France

[§]Inria, Montpellier - France

[¶]I2BC (Institute for Integrative Biology of the Cell), CEA, CNRS, Univ. Paris-Sud - France

[♯]DIRO (Département d'Informatique et de Recherche Opérationnelle) - Univ. Montréal - Québec - Canada

[⋆]Corresponding authors

Proceedings of the VLDB Endowment, Vol. 8, No. 11

1. INTRODUCTION

The problem of aggregating multiple rankings into one consensus ranking has started to be investigated two centuries ago and has been actively studied again in the last decades. Direct applications are numerous and include aggregating answers returned by several web engines [20], computing a global rating based on numerous user ratings [4, 33], determining the winner in a sport competition [5], or combining biomedical orderings [12, 18, 32].

This topic has been of particular interest in the information retrieval and database communities ([20] and [21, 22, 23, 27, 31, 34]) while several other communities have also deeply looked into it, including algorithmics ([1, 2, 31]), artificial intelligence ([5]), and social sciences ([3]).

Various studies have implemented methods for rank aggregation and may have come up with contradicting conclusions upon which algorithms work best. Comparing such results is cumbersome, as the original studies mixed different kinds of approaches and used very different evaluation datasets and metrics.

Additionally, in real applications, the rankings to be aggregated may not be permutations where elements are strictly ordered, but they may have ties where some elements are placed at the same position (*e.g.*, *ex-aequo* competitors, or ratings involving the same grade to be possibly associated with several elements). While the first efficient solution to rank aggregation considering input rankings with ties has been introduced in 2004 [21], most of the approaches and studies introduced since then have continued to focus on permutations, leaving several open questions in the context of ranking with ties.

The purpose of this paper is thus twofold: (i) it provides a clear overview of the approaches able to aggregate rankings with ties and (ii) it introduces the first complete study on ranking with ties, including possible (or not) adaptation of existing algorithms to deal with ties, and experimentation of the approaches both on real and carefully generated synthetic datasets.

More precisely, this paper makes four contributions. First, we provide a comprehensive review of the existing rank aggregation approaches and a clear overview of the results provided in the literature. Second, we carefully study the impact of considering ties in all approaches, both on exact and approximation (or heuristics) solutions, and provide a new translation into integer linear programming for finding an optimal consensus ranking in the context of ties. Third,

we present the results we obtained on real and synthetic datasets by using available rank aggregation algorithms. Finally, a careful analysis of the results allows us to provide a comprehensive view of the algorithms to be used depending on the kind of application and carefully identified *datasets features* (e.g., number of elements, similarity between input rankings...).

The paper is organized as follows. After introducing the formal background in Section 2, we review the major rank aggregation algorithms available in the literature (Section 3). In Section 4, we study the consequence of considering ties both on exact and approximation (or heuristics) solutions. Previous results on the performance and relative quality of existing approaches are summarized in Section 5. Section 6 describes the experimental setting of the study, the datasets we gathered and/or generated, and the methodology used for comparing the available approaches. We analyze our results and provide guidance on the algorithm to be favored based on dataset features in Section 7 while we discuss perspectives in Section 8.

2. BACKGROUND

2.1 Rank Aggregation problem

Among the communities the rank aggregation problem is named differently including *Kemeny rank aggregation* [3, 5, 6, 14], *consensus ranking* [30], *median ranking* [12], and *preference aggregation* [17].

The rank aggregation problem has been originally defined for a set of permutations. A **permutation** π is a bijection of $[n] = \{1, 2 \ldots, n\}$ onto itself. It represents a strict total order of the elements of $[n]$ (it is thus a *ranking*). The set of all permutations of $[n]$ is denoted $\mathbb{S}_n$, and its size is $|\mathbb{S}_n| = n!$. As usual, $\pi[i]$ stands for the image (or position) of integer i in permutation π, and we denote $\pi = \pi[1]\pi[2] \ldots \pi[n]$.

A classical dissimilarity measure for comparing two permutations is the **Kendall-τ distance** [29] which counts the number of pairs for which the order is different in the two permutations. More formally, the Kendall-τ distance, here denoted D, counts the pairwise disagreements between two permutations, π and $\sigma \in \mathbb{S}_n$, and is defined as:

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

Other metrics exist (*e.g.*, Spearman's footrule [19]), which are all within constant multiples of each others [21].

From the Kendall-τ distance, the **Kemeny score** [28] is defined as the sum of Kendall-τ distances between a given permutation and all permutations in a given set. More formally, given any set of permutations $\mathcal{P} \subseteq \mathbb{S}_n$ and a permutation π, we define the Kemeny score S as:

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

Finally, the problem of finding an optimal consensus π^* of a set $\mathcal{P} \subseteq \mathbb{S}_n$ is to find π^* such that:

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

The optimal consensus π^* is also denoted as *optimal Kemeny ranking* [1, 3, 4, 5], *optimal aggregation* [20], *optimal solution* [2, 27, 30, 31] and *median* [12].

There may be several optimal consensus.

Example: Let us consider the set of input permutations $\mathcal{P} = \{\pi_1, \pi_2, \pi_3\}$ where $\pi_1 = [A, D, B, C]$, $\pi_2 = [A, C, B, D]$, $\pi_3 = [D, A, C, B]$. The optimal consensus ranking of $\mathcal{P}$ is $\pi^* = [A, D, C, B]$. The Kemeny score of π^* is made of the pairwise disagreements A-D in π_3, D-C in π_2, D-B in π_2, and C-B in π_1 thus $S(\pi^*, \mathcal{P}) = 4$.

Considering the Kendall-τ distance, the rank aggregation problem is known to be NP-hard when the number of permutations in $\mathcal{P}$ is even and greater or equal to 4 [7, 20]. It is an open problem when the number of permutations is odd.

2.2 Ranking with ties

We now consider the problem of rank aggregation with ties, that is, when elements in the input rankings can be tied (with equal rank). More formally, following [21], a **bucket order** on $[n]$ is a transitive binary relation $\lhd$ represented by a set of non empty buckets $\mathcal{B}_1, \ldots, \mathcal{B}_k$ that forms a disjoint partition of $[n]$ such that $x \lhd y$ if and only if there are i, j with $i < j$ such that $x \in \mathcal{B}_i$ and $y \in \mathcal{B}_j$. A **ranking with ties** on $[n]$ is defined as $r = [\mathcal{B}_1, \ldots \mathcal{B}_k]$, where $r[x] = i$ iff $x \in \mathcal{B}_i$. Although the classical formulation of the Kendall-τ distance allows to compare rankings with ties [3, 31], in this case it is not a distance anymore [22]; moreover, ties are actually ignored and no disagreement can be considered for (un)tied elements. Therefore, whatever the input is, the ranking with the fewest disagreements is the ranking where all elements are tied in a unique bucket. To avoid producing such a non-sense solution, algorithms based on Kendall-τ have to restrain themselves to produce permutations. As a consequence, the **generalized Kendall-τ distance**, denoted G, has been introduced to address the need of producing ranking with ties. It is defined as follows:

$$G(r, s) = |\{ \ (i,j) : i < j \ \wedge$$
$$((r[i] < r[j] \ \wedge \ s[i] > s[j]) \ \vee \ (r[i] > r[j] \ \wedge \ s[i] < s[j]) \ \vee$$
$$(r[i] \neq r[j] \ \wedge \ s[i] = s[j]) \ \vee \ (r[i] = r[j] \ \wedge \ s[i] \neq s[j])) \ \}|$$

A pair of elements which are either inversed or tied in only one ranking counts as one disagreement. Computing the distance is equivalent to sorting the elements and can be done, with adaptations, in log-linear time when considering ranking with ties [20].

[10, 12, 21] assign a different cost to the case where two elements are inversed compared to the case where two elements are tied in only one ranking. Here, we consider a cost of one in both cases.

Let $\mathcal{R}_n$ be the set of all possible rankings with ties over [n]. Given any subset $\mathcal{R} \subseteq \mathcal{R}_n$ and a ranking r, the **generalized Kemeny score** denoted K is :

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

An optimal consensus ranking of a set of rankings with ties $\mathcal{R} \subseteq \mathcal{R}_n$ under the generalized Kemeny score is a ranking with ties r^* such that

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

A **consensus ranking** (or consensus for short) denotes a not necessarily optimal solution of the problem. When a solution is optimal it is explicitly denoted as an **optimal consensus**.

Example: Let us consider the set of input rankings $\mathcal{R} = \{r_1, r_2, r_3\}$ where $r_1 = [\{A\}, \{D\}, \{B, C\}]$ (B and C are

Ref	Name	Approx.	Algorithm class	Can produce ties	Untying cost
[1]	**Ailon$\frac{3}{2}$**	3/2	[K] Linear Prog.	with slight modification	with slight modif.
[12]	**BioConsert**	2	[G] Local search	yes	yes
[8],[16]	**BordaCount**	5	[P] Sort by score	with slight modification	no
[11]	Chanas	no	[K] Local search	no	—
[13]	ChanasBoth	no	[K] Local search	no	—
[3]	BnB	exact	[K] Branch & Bound	no	—
[15]	**CopelandMethod**	no	[P] Sort by score	with slight modification	no
[21]	**FaginDyn**	4	[G] Dynamic Prog.	yes	yes
[3, 5, 14, 31]	**ILP**	exact	[K] Linear Prog.	only in input	with large modif.
[2]	**KwikSort**	$\frac{11}{7}$	[K] Divide & conquer	with slight modification	with slight modif.
[20]	MC4	no	[P] Hybrid	yes	no
[24]	**MEDRank**	no	[P] Extract order	with slight modification	no
[2]	**Pick-a-Perm**	2	[K] Naive	yes	—
[1]	**RepeatChoice**	2	[K] Sort by order	with slight modification	no

Table 1: Algorithms and their categories, "Approx." stands for approximation, "[P]" for Positional algorithms, "[K]" for Kendall-τ based algorithms and "[G]" for generalized Kendall-τ based algorithms. Algorithms in bold have all been re-implemented and experimentally evaluated (cf. Section 6).

tied), $r_2 = [\{A\},\{B,C\},\{D\}]$, $r_3 = [\{D\},\{A,C\},\{B\}]$. The optimal consensus of $\mathcal{R}$ is $r^* = [\{A\},\{D\},\{B,C\}]$. The generalized Kemeny score $K(r^*,\mathcal{R})$ is made of inversions *A*-*D* in r_3, *D*-*B* in r_2, *D*-*C* in r_2, tying *B*-*C* in r_3 and untying *A*-*C* in r_3 thus $K(r^*,\mathcal{R}) = 5$.

In this paper, a **dataset** systematically denotes a set of input rankings $\mathcal{R}$.

3. CLASSIFICATION OF APPROACHES

We review rank aggregation algorithms. We distinguish approaches on whether their objective function is focused on the disagreements regarding the order of pairs of elements, considering (i) the generalized Kendall-τ distance ([G] in Table 1), or (ii) the (classical) Kendall-τ distance ([K]), or focused on the position of elements in rankings ([P]).

3.1 Generalized Kendall-τ based algorithms

We describe here the only two approaches designed to natively deal with ties.

FaginDyn [21] is a **dynamic programming** approach which runs in time $\mathcal{O}\left(nm + n^2\right)$ (n elements among m rankings). Variants can be considered, favouring solutions with large (*FaginLarge*) or small (*FaginSmall*) buckets [12].

BioConsert [12] follows a **local search** approach. It classically starts from a solution (*i.e.* a ranking) and applies edition operations to alter the solution as long as the cost of the current solution is reduced. The two edition operations in *BioConsert* are (i) removing an element from its current bucket and placing it into a new bucket at a given position and (ii) moving an element in an already existing bucket. *BioConsert* has an $\mathcal{O}\left(n^2\right)$ memory complexity.

3.2 Kendall-τ based algorithms

We now review approaches based on the Kendall-τ distance. They can take rankings with ties as input but (by definition of the distance) ignore the cost of (un)tying and produce permutations as output (cf. Section 2.2).

The rank aggregation problem has naturally been translated into an **integer linear programming** problem (ILP) and several strategies proposed to minimize the complexity of solving the ILP [3, 14, 31]. A polynomial preprocessing is introduced in [5, 6] to divide the problem into smaller instances. Ailon [1] presents a $\frac{3}{2}$-approximation (called *Ailon*$\frac{3}{2}$ here) based on relaxing the ILP into a floating-point optimization problem. The reconstruction of the consensus ranking is then achieved by rounding variables to integers.

KwikSort[2] is a $\frac{11}{7}$-approximation algorithm based on a **divide-and-conquer** approach (the $\frac{11}{7}$ bound holds when choosing the best of KwikSort and Pick-a-Perm solutions). Given a set of elements, it (recursively) randomly chooses a pivot and assigns the other elements to two buckets placed before and after the pivot so that each element minimizes the number of pairwise disagreements with the pivot. A de-randomized version of KwikSort has been studied on a theoretical point-of-view [35]. KwikSort outperforms all other approaches based on sorting [31]. Its memory consumption is at worst pseudo linear in n.

In [3] a **branch-and-bound** approach explores a tree where each leaf at depth j represents a part of the solution over the j^{th} first elements such that the currently studied leaf has the minimal number of disagreements. It can return optimal solutions. Heuristics are proposed based on techniques limiting the number of leaves expended.

Chanas [11] and *ChanasBoth* [13, 31] are two greedy **local search** approaches where the edition operation permutes two consecutive elements.

Other Kendall-τ based algorithms consist in Pick-a-Perm and RepeatChoice. *Pick-a-Perm* [2] is a naive approach which takes permutations as input and returns one of the input rankings. A de-randomized version [31] returns one input ranking with minimal cost.

RepeatChoice [1] is a 2-approximation (called *Ailon2* in [12]) derived from Pick-a-Perm which provides permutations. Starting from one input ranking, the buckets are broken following the order of the elements in the other input rankings (randomly picked) until all input rankings have been used. Remaining buckets are arbitrarily broken. A simple implementation runs in $\mathcal{O}\left(m \times S(n)\right)$.

3.3 Positional algorithms

Approaches described here make use of the position of elements to produce rank aggregation. They all have been

designed to produce permutations as output (*i.e.* rankings without ties). Some approaches follows a **Voting scheme**. In *BordaCount* [8], the position of an element is defined as the number of elements placed before it, plus one. The score of an element is then the sum of its positions in the rankings. In *CopelandMethod* [15] the score is the sum of the number of elements placed after it. Both methods run in $\mathcal{O}(nm + S(n))$ where $S(n)$ is the complexity of the sorting algorithm, n the number of elements, and m the number of input rankings.

MEDRank [24] is a fast algorithm considering a *Top-k aggregation* strategy to avoid any sorting step: input rankings are read in parallel, element-by-element. Given m rankings and a threshold in $h \in]0;1[$, as soon as an element has been read in $h \times m$ rankings, it is appended to the consensus. It runs in $\mathcal{O}(nm)$.

A last kind of approaches (qualified as **hybrid** in [31]) includes *MC4* [20] where the problem of rank aggregation is represented by a Markov chain whose states are elements in the input rankings. The probability of a transition between two elements e_1 and e_2 is equal to $\frac{1}{n}$ if there is a majority of rankings preferring e_2 to e_1 (n is the number of elements). The score of each element is its probability in the stationary distribution of the chain. Elements are then sorted in ascending order. The complexity of this method is dominated by the complexity of finding the stationary distribution.

4. IMPACT OF TIES

The complexity of the problem of ranking aggregation with ties has not been considered so far. Permutations can be seen as rankings with ties where each bucket is of size one. Considering a set of such rankings, we have proved that under the generalized Kendall-τ distance (cf. Section 2.2) the optimal consensus obtained has necessarily only buckets of size one [9]. Thus the problem of ranking aggregation under the Kendall-τ distance is a particular case of the problem of ranking with ties under the generalized Kendall-τ distance. The complexity result of [7, 20] thus still holds: the problem of aggregating ranking with ties is NP-hard when m the number of rankings is even and $m \geq 4$ and is open when m is odd. Designing approximations and heuristics is thus still of paramount importance in this context.

Here, we first adapt (whenever possible) existing algorithms to make them consider ties and assign a cost to untying elements. We then present a new translation of the problem to an integer linear programming to obtain an exact solution able to consider the cost of untying elements.

4.1 Adapting existing approaches

We discuss a general methodology to adapt ranking algorithms to ties and provide such adapted algorithms. Algorithms described in Section 3.1 natively deal with ties, do not need any adaptation, and thus are not considered here.

4.1.1 Methodology to adapt algorithms to ties

Algorithms considering the Kendall-τ distance (cf. Section 3.2) can be adapted to produce rankings with ties following three different strategies. First, for algorithms based on branching depending of placing element(s) before or after another, the presence of ties brings a third choice: putting them in the same bucket. It can either result in an adaptation of the original algorithm (KwikSort), or as a new algorithm (BnB). Second, for algorithms based on local search (e.g., Chanas), new operations have to be designed to explore the search space, and can result as a new algorithm (BioConsert). Third, for linear programming algorithms, a new formalism has to be drawn (cf. Section 4.2).

As for algorithms using the position (cf. Section 3.3) of elements to compute a consensus, they can directly be used with ranking with ties as the formulation of the position of an element encompasses the presence of ties. Considering the cost of (un)tying elements is not directly possible and implies designing ad-hoc solutions.

4.1.2 Adapting Kendall-τ based approaches

Local search: Neither Chanas nor ChanasBoth handle ties. However, BioConsert is a local search approach designed to handle ties and considers the cost of untying elements.

Divide and conquer: The formulation of KwikSort can be adapted to encompass ties. Elements should have the possibility to be tied to the pivot in addition to be placed before or after the pivot. Minimizing the pairwise disagreements now includes the cost of (un)tying elements. The complexity is modified by a constant factor only.

Branch-and-bound: The BnB algorithm has been designed for permutations only. Dealing with rankings with ties would require designing a fully new algorithm.

Linear programming: The Ailon $\frac{3}{2}$ approach relaxes the problem in floating-point optimization and can be used as it is. As for optimal solution, a new algorithm is introduced in Section 4.2.

Other: Pick-a-Perm can be used directly with ties. RepeatChoice takes in rankings with ties and produces a permutation by arbitrarily breaking the possible remaining ties. Removing this last step makes the algorithm able to produce rankings with ties.

4.1.3 Adapting positional algorithms

BordaCount and CopelandMethod can be adapted by following the general methodology introduced above. There is no change in their complexity. However, they are not able to consider the cost of (un)tying elements. As an example, let us consider two elements x and y, among others, ranked in a set of input rankings. If in one input ranking x and y are not tied while in all other input rankings they are tied, then their scores will be different. As a consequence, x and y will be untied in the consensus provided by BordaCount or CopelandMethod, although a very large majority of the input rankings considers them as "equivalent" (*i.e.* tied).

MEDRank can easily be adapted to ties (in one ranking, multiple elements can be read at the same time if they are tied in a bucket) without any change in its complexity.

The hybrid approach MC4 uses a graph-based representation and may take rankings with ties as input as it models the order of elements with edges. Nevertheless, this approach is costly and considering the cost of (un)tying elements would imply considering a different Markov chain modeling.

4.2 A ties-aware optimal algorithm

Here we generalize the existing method of [14] to ties. Our key insight is to express our problem as a linear pseudo-boolean optimization problem (LPB), *i.e.*, as a linear program whose variables are pseudo-boolean: their values are

either zero or one, and they can be subject to classical arithmetic operations. In a LPB problem, the objective is to find an assignment of boolean variables such that all constraints are satisfied and the value of the linear objective function is optimized. Considering Linear Programming is natural and has been already done for dealing with permutations. [14] provides an LP formulation using a set C of candidates, some weight coefficients $w_{a<b}$ counting for the number of input rankings having a appearing before b and some binary variables $x_{a<b}$ assessing if, in the optimal consensus ranking, a appears before b. The objective function is to minimize the disagreements *i.e.* $\sum_{\{a,b\}\subseteq C} (w_{b<a} * x_{a<b} + w_{a<b} * x_{b<a})$ while respecting the following two constraints: i) $\forall\{a,b\} \subseteq C, x_{a<b} + x_{b<a} = 1$ and ii) $\forall\{a,b,c\} \subseteq C, x_{a<c} - x_{a<b} - x_{b<c} \geq -1$. Constraint (i) ensures that any two elements are uniquely ordered in the optimal consensus ranking and (ii) ensures order transitivity.

Our LPB formulation deals with rankings with ties. We make use of the notations of [14]. We now present the variables, the objective function and the constraints of our LPB program.

Variables. For any pair of elements (a,b), we define a variable $x_{a<b}$ to denote whether, in the optimal consensus ranking, a is ranked before b. Since in a ranking with ties two elements may be unordered, we also introduce a variable $x_{a=b}$ to denote whether in the optimal consensus ranking, elements a and b are in the same bucket. As before, $w_{a<b}$ counts for the number of input rankings where a appears before b, while $w_{a\leq b}$ counts for the number of input rankings where a appears before or in the same bucket as b.

Objective. The objective of the LPB program is the expression of the generalized Kendall-τ distance using variables of our LPB problem:

$$\sum_{a,b}(w_{b\leq a} * x_{a<b} + w_{a\leq b} * x_{b<a} + (w_{a<b} + w_{a>b}) * x_{a=b})$$

Clearly, if a appears before (resp. after) b in the optimal consensus ranking, any input ranking where a appears after (resp. before) b or tied to b costs one. Moreover, any pair of elements a and b in the optimal consensus ranking which are tied costs one for any input ranking where they are not.

Constraints. We now add constraints to ensure that the solution returned is a ranking with ties. First, we generalize constraint (i) above to ensure that any two elements are uniquely ordered in the optimal consensus ranking: either a is ranked before b, or b is ranked before a or a and b are in the same bucket. Thus, we must have:

$$x_{a<b} + x_{b<a} + x_{a=b} = 1 \quad (1)$$

In order to ensure order transitivity, we have the same constraint as (ii) above, *i.e.* if a is ranked before b and b is ranked before c then a is ranked before c:

$$x_{a<c} - x_{a<b} - x_{b<c} \geq -1 \quad (2)$$

Finally, we ensure that the solution is a ranking with ties. Roughly, if a and b are in the same bucket and so do b and c, then all three of them are in the same bucket.

$$2x_{a<b} + 2x_{b<a} + 2x_{b<c} + 2x_{c<b} - x_{a<c} - x_{c<a} \geq 0 \quad (3)$$

LEMMA 1. *Our LPB program correctly solves the problem of finding an optimal consensus ranking under the generalized Kendall-τ distance of a set of rankings with ties.*

PROOF. Let us first prove that a solution to the rank aggregation problem can be found by our LPB program *i.e.* that it has a corresponding variable assignment that respects all the constraints previously defined. Given an optimal consensus ranking r^* and for each pair of elements (a,b), either a is ranked before b ($x_{a<b} = 1$), or b is ranked before a ($x_{b<a} = 1$) or a and b are in the same bucket ($x_{a=b} = 1$). Thus, our assignment ensures that $x_{a<b} + x_{b<a} + x_{a=b} = 1$ (Constraint (1)) for any pair of elements (a,b). Since r^* is a ranking with ties, it ensures transitivity. Considering a triple of elements (a,b,c) where a is before b ($x_{a<b} = 1$) and b is before c ($x_{b<c} = 1$). The only way to disregard Constraint (2) would then to have $x_{a<c} = 0$ leading to a contradiction since a is before c by transitivity. Finally, considering Constraint (3), clearly, if any of $(x_{a<b}, x_{b<a}, x_{b<c}, x_{c<b})$ is set to one, then the constraint is satisfied. Consider then that $x_{a<b} = x_{b<a} = x_{b<c} = x_{c<b} = 0$, then it means that a and b are tied and so are b and c. By definition, a, b and c belong to the same bucket, thus $x_{a<c} = x_{c<a} = 0$.

Let us now prove that a solution to our LPB program corresponds to a ranking with ties. Given any solution, Constraint (1) ensures that for any pair of elements a and b, either a is ranked before b if $x_{a<b} = 1$, or a is ranked after b if $x_{b<a} = 1$, or a and b belong to the same bucket if $x_{a=b} = 1$. Moreover, Constraint (2) ensures the transitivity of the ranking. Finally, Constraint (3) ensures that the resulting ranking is a ranking with ties.

The objective function fully corresponds to the computation of the generalized Kendall-τ distance, thus an optimal solution to our LPB is an optimal consensus ranking. □

Due to the intrinsic complexity of this linear problem, optimal solutions can be computed for moderately large datasets only. This allows us evaluating the quality of non-optimal algorithms.

5. PREVIOUS FINDINGS

We now summarize the results obtained by previous studies on the performance of rank aggregation algorithms. We start with describing the datasets (listed in Table 2) and the **normalization processes** which are used. Results on real and synthetic datasets are then presented.

5.1 Normalization Process

Studies have designed approaches to convert any dataset (i.e, a set of rankings) over different elements into dataset over the same elements. Normalization strategies are described here and illustrated in Table 3.

Projection consists in removing from a dataset all the elements which are absent in at least one ranking [5], resulting in a ***projected dataset***. In Table 3 d_p is the projection of d_r. It always produces permutations when the input rankings are permutations. Its major drawback is to possibly remove (large sets of) relevant elements.

Unification consists in adding at the end of each ranking of a dataset a **unification bucket** with elements appearing in other rankings (and absent in the current ranking). This last unifying bucket can be considered as is (as in [12, 31]), we say that such datasets are ***unified***, d_u is the unification of d_r. Others [3] prefer to arbitrarily break this bucket to consider only permutations in the input rankings, such datasets are said to be ***unif[ied] broken***, as dataset d_b in Table 3 .

Name & publications	ava-ila-ble	Over the same elements	Used with ties
EachMovie [13]		no	no
F1 [5]	[5]	projected	no
BioMedical [12]	[12]	unified	yes
GiantSlalom [3]		unif. broken	no
SkiCross/Jumping [5]	[5]	projected	no
WebCommunities [13, 31]		yes	no
WebSearch [20, 31, 3, 5]	[5]	unified[31, 3] projected[5]	yes no
Mallows model [3, 5]		yes	no
Placket-Luce model [3, 5]		yes	no
RandomGraph [17, 13]		yes	no
Random [3]		yes	no
Random [12]		yes	yes

Table 2: Datasets properties and availability.

Raw dataset d_r	Projected dataset d_p
$[\{A\},\{D\},\{B\}]$	$[\{A\},\{B\}]$
$[\{B\},\{E,A\}]$	$[\{B\},\{A\}]$
$[\{D\},\{A,B\},\{C\}]$	$[\{A,B\}]$
Unified dataset d_u	Unif. broken dataset d_b
$[\{A\},\{D\},\{B\},\{C,E\}]$	$[\{A\},\{D\},\{B\},\{C\},\{E\}]$
$[\{B\},\{E,A\},\{C,D\}]$	$[\{B\},\{A\},\{E\},\{C\},\{D\}]$
$[\{D\},\{A,B\},\{C\},\{E\}]$	$[\{D\},\{A\},\{B\},\{C\},\{E\}]$

Table 3: Resulting datasets after applying the various normalization processes to the *raw dataset* d_r.

5.2 Previous results

In this subsection, we provide a view of the results obtained by the previous studies found in the literature.

The only experimental study which has evaluated approaches both taking in and producing rankings with ties is [12]. They concluded their experimentations by stating that BioConsert outperformed all other considered approaches. However, results have been obtained on small real (biomedical) datasets and on very small generated datasets (4 to 8 elements). Both the algorithms and the datasets considered in [12] totally differ from those used in other studies [3, 5, 13, 31].

Other studies ([3] and [31]) have compared approaches taking in rankings with ties but based on the classical Kendall-τ distance, thus necessarily producing permutations and unable to consider the cost of (un)tying. [3] work on WebSearch and GiantSlalom datasets while [31] work on WebSearch and WebCommunities datasets. From their respective experimentations they conclude that (i) Chanas produces good quality results, (ii) positional approaches (such as BordaCount and CopelandMethod) are approaches able to quickly provide results of good quality, (iii) KwikSort provides a good trade-off between the previous recommendations. Additionally, [3] recommends to use BnB with beam search techniques as an intermediate solution between KwikSort and ChanasBoth. While in [3] CopelandMethod returns better results than BordaCount, in [31] they appear to be equivalent. As for CopelandMethod and MC4, in [31] they present comparable results in term of quality, MC4 being much more time consuming.

Another study, [13], focuses on permutations only, using WebCommunities and EachMovie datasets. Compared to [3, 31], they confirm that Chanas and ChanasBoth produce quality results but they fully disagree on the use of KwikSort (which obtains bad performance). Positional algorithms are not considered at all in this study.

Optimal solutions (ILP-based) are intensively studied by [5] which introduces pre-processes to reduce the search space of the problem.

In addition to real datasets, a few generated datasets have been considered. In particular and interestingly, [3] generated some datasets with different levels of similarities. BordaCount appears to be the best choice while BnB should be preferred when the similarity is low. However, as authors do not provide any means to determine the similarity between given input rankings, such recommendations remain difficult to apply in concrete settings.

As for the normalisation process, [3, 31, 12] normalize datasets with the unification process, while [5] project them. Whether or not the normalization process may have an impact on their results is an open question.

To summarize, each study has considered a given (restricted) set of algorithms and has performed experimentations in very different datasets, curated with diverse methods, and producing mostly permutations. Given the current results that can be extracted from the literature, it is thus very difficult to determine in which context one approach should be preferred over the others.

The next two sections introduce for the first time the results obtained on a large-scale study conducted on rank aggregation algorithms considering ties.

6. EXPERIMENTAL SETTING

In this section, we describe both the datasets we used to compare rank aggregation algorithms, and the methodology followed in our experiments.

6.1 Datasets

We have first considered real-world datasets which have already been used in previous works (the four groups of datasets in bold in Table 2) while extending the experiments to a very large set of algorithms. We have then carefully generated datasets to better understand the possible impact of three dataset features on rank aggregation algorithms: the size of the dataset (number of rankings, number of elements in each ranking), the (level of) similarity between the rankings taken as input, the normalization process which has been applied to the dataset (unification or projection). The generation of synthetic datasets is described in the next two subsections.

6.1.1 Uniformly generated synthetic datasets

The datasets introduced here are made of rankings with ties, where all rankings have the same probability to be present. This has been carefully ensured by using the MuPAD-Combinat package [26] based on the work of [25].

More precisely, we produced datasets of $m \in [3; 10]$ rankings for different lengths: $n \in [5; 95]$ with a step of 5, and $n \in [100; 500]$ with a step of 100. We produced 100 datasets for each pair $< m, n >$. Such numbers have been chosen to mimic real-world settings.

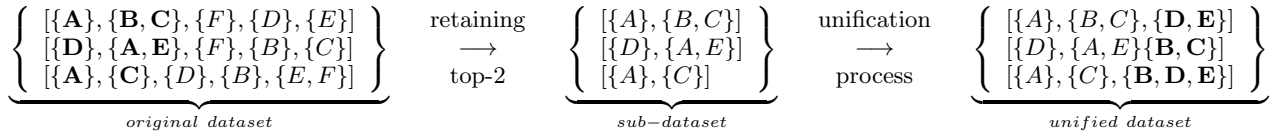

Figure 1: Generation of a unified synthetic dataset: a dataset over 6 elements is generated (100 in the experiment), then only the top-k=2 elements are retained for each ranking (k∈[1;35] in the experiment). The unification process is finally applied to obtain datasets over the same elements.

6.1.2 Increasingly dissimilar synthetic datasets

To study the impact of input rankings similarity on the results obtained, we modeled the data generation process by a Markov chain where states are rankings with ties and transition between states represents a possible modification of one ranking into another. Modifications are done using four operators: move an element of a ranking in the previous or the following bucket, put it in a new bucket right before or right after its current position. Such operators ensure (with restrictions when buckets contain one or two elements, details omitted) that the Markov chain converges to the uniform stationary distribution. Considering a seed ranking r_s and a number of steps t, a dataset over m rankings consists in starting m times from r_s in the Markov chain and adding the state currently visited after t steps. The modeling allows us to generate rankings with ties biased by the starting state, and with different levels of similarity to the seed ranking (depending of the number of steps allowed in the Markov chain), and thus different levels of similarity.

We generated 1000 datasets of $m = 7$ rankings over $n = 35$ elements with a number of steps to walk in the Markov chain process $t \in \{50, 100, 250, 500, 1000, 2500, 5000, 10000, 25000, 50000^1\}$.

6.1.3 Unified synthetic datasets with similarities

Here, we study the impact of the unification process (introduced in Section 5.1) used either when datasets are not over the same elements (BioMedical [12]), or when the dataset is made of only the top-k elements of raw rankings (WebSearch [20, 31]). We mimicked this second use case and generated datasets with different levels of similarity (cf. Section 6.1.2), retained only the top-k elements and then applied the unification process (cf. Figure 1).

We generated 1000 datasets with $m = 7$ rankings over $n = 100$ elements with a number of steps t to walk in the Markov chain modeling, $t \in \{10^3, 2.5 * 10^3, 5 * 10^3, 10^4, 2.5 * 10^4, 5 * 10^4, 10^5, 2.5 * 10^5, 5 * 10^5, 10^6\}$. The top-k elements are retained with $k \in [1; 35]$ in order to have datasets of $n = 35$ elements.

6.2 Methodology

6.2.1 Algorithms

Algorithms we have entirely re-implemented and evaluated are indicated in bold in Table 1. To evaluate randomized algorithms and highlight the variability of the results quality of such algorithms, we have considered a large number of runs and selected as solution the best solution computed through the iterations. We thus present the results with the name of the algorithm suffixed with "Min": *RepeatChoiceMin*, *KwikSortMin*.

[1]This maximum number of steps is discussed in Section 7.2.

6.2.2 Measuring similarity

The Kendall$-\tau$ rank correlation coefficient [29] is a well-known measure to quantify the correlation of two permutations. Its extension to rankings with ties is straightforward. Formally, considering r_1, r_2 two rankings with ties, it is defined as:

$$\tau(r_1, r_2) = \frac{\frac{1}{2}n(n-1) - 2G(r_1, r_2)}{\frac{1}{2}n(n-1)} \quad (4)$$

where G is the generalized Kendall-τ distance (cf. Section 2.2). To measure the intrinsic correlation of a dataset $\mathcal{R} = \{r_1, ...r_m\}$, we average the correlation coefficient of each pair of rankings in this dataset:

$$s(\mathcal{R}) = \frac{2}{m(m-1)} \sum_{i=1}^{m} \sum_{j=i+1}^{m} \tau(r_i, r_j) \quad (5)$$

6.2.3 Quality of the results

To compare the quality of the results two approaches may be followed. The first one is to use the actual distance of each consensus to the inputs rankings [13]. The second approach named *gap* [3, 31] consists in normalizing the distance to show the additional disagreement a solution has, compared to an optimal solution. Formally, let c^* be an optimal consensus ranking and c be a consensus ranking returned by a given algorithm for a set of rankings with ties $\mathcal{R}$, the *gap* is defined such that:

$$gap = \frac{K(c\ , \mathcal{R})}{K(c^*, \mathcal{R})} - 1 \quad (6)$$

As a consequence, optimal consensus have a *gap* of 0.

In the case where it was not possible to compute any optimal solution, we compute the $m-gap$, where the distance of a result produced by an algorithm is normalized by the distance of the best consensus proposed by any available algorithm.

6.2.4 Comparing over time

We paid particular attention to the configuration we set up to ensure fair time comparison. Experiments were conducted on a four dual-core processor Intel Xeon 3GHz with 16GB memory using Java 1.6.0_37, LPSolve 5.5.2.0, CPLex 12.4 and Python 2.4.4. To measure the execution time of an algorithm, we ran it numerous times in a row such that the time needed to do all executions was greater than two seconds (the execution time is then the overall execution time divided by the number of executions). Each measure was preceded by a warm-up time to ensure that all classes were already loaded in the JVM memory. Implementations were single-threaded. LPSolve (used by Ailon$\frac{3}{2}$) and CPlex (used by the ExactAlgorithm) were used in their default configuration. For every algorithm we limited the computing time

Algorithm	WebSearch Unif (%) Proj	WebSearch Unif ($m-gap$, %)	F1 (%)Proj	F1 (%)Unif	SkiCross (%) Proj	SkiCross (%)Unif	BioMed. (%)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%
CopelandMethod	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%
KwikSortMin	0,2(# 3)	32,2(# 2)	0(# 3)	0,2(# 2)	1,8(# 2)	15,1(# 3)	0,16(# 1)	61,0%
MEDRank(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(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)	
RepeatChoice	24,8(#12)	57,5(#12)	27,5(#13)	54,5(#13)	23,3(#12)	34,6(#12)	31,9(#12)	
RepeatChoiceMin	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

Table 4: Average gap (m-gap for unified WebSearch datasets) obtained over all datasets, and their rank.

to two hours: after that limit, we considered that the algorithm was not able to provide a solution.

7. RESULTS

In this section, we present the results obtained by a large panel of rank aggregation algorithms using various kinds of datasets. Firstly, we evaluate the quality of results produced by the algorithms and their time consumption while considering only the size of the datasets, to this end we use uniformly generated datasets. Secondly, we take into account different levels of similarity with the help of synthetic datasets with progressive dissimilarity. Thirdly we analyze the impact of the normalization processes (unification, projection) on real datasets. Last, we analyze the impact of the unification process on the quality of the results produced.

In each analysis, we also study the impact of results obtained on real world datasets with the same knowledge (size, similarity, normalization) allowing us to highlight the information needed to fully understand the behaviour of algorithms in real settings.

7.1 Considering only the size of datasets

7.1.1 Measuring quality

We have conducted a first series of experiments on uniformly generated datasets and observed that the number of rankings considered has no significant impact in the results. We have systematically considered datasets with $m \in [3; 10]$ rankings and $n \leq 60$ elements (60 is the highest number of elements for which the optimal consensus ranking can be computed in a reasonable amount of time).

Results are presented in Table 5. Four points deserve attention.

First, for both synthetic and real datasets (cf. Table 4 and 5), BioConsert provides the best results in the very large majority of the cases (88.56%) and in 33.94% it strictly outperforms the other algorithms. In 68.01% of the cases, the solutions produced are optimal. When focused on real datasets only, BioConsert provides the best results in 91.8% of the datasets.

Second, Ailon$\frac{3}{2}$ is a very effective approximation since it has a *gap* close to 0, meanwhile the current implementation of the algorithm does not scale: for $n > 45$ no result is provided. On synthetic datasets, Ailon$\frac{3}{2}$ and BioConsert provide results with a similar quality: Ailon$\frac{3}{2}$ strictly outperforms BioConsert in 14.39% of the datasets, while it is outperformed in $18,67\%$ of them.

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

Table 5: Average *gap* (and rank), percentage of datasets where the optimal consensus ranking is found and percentage of datasets where the algorithm is first on uniformly generated datasets over $n \leq 60$ elements.

Third, positional algorithms BordaCount and CopelandMethod have an interesting behavior on synthetic datasets. When being compared to the other algorithms, the average disagreements of their solution surprisingly decreases as the number of elements grows: BordaCount (resp. CopelandMethod) is ranked 8^{th} (resp. 9^{th}) when considering datasets of 20 elements, and 3^{rd} (resp. 4^{th}) with datasets of 500 elements.

Fourth, we have evaluated the behavior of MEDRank when varying its threshold. General observations of the *gap* for various values of the threshold have shown that MEDRank is very sensitive to its threshold value. More precisely, values higher than the default one (threshold of 0.5) do not lead to any improvement in the quality of the consensus provided, neither in real nor in synthetic datasets. In 76.37% of the synthetic datasets a threshold of 0.5 provides the best results. It is thus the threshold value

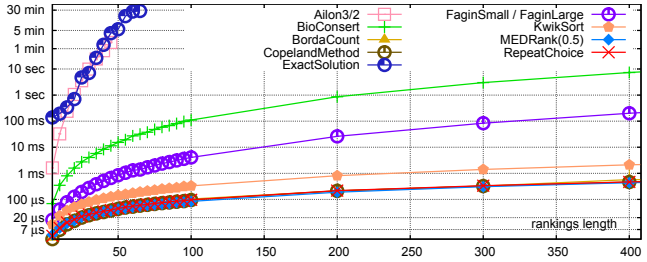

Figure 2: Computing time with $n \in [5; 400]$. Note that BordaCount, CopelandMethode, MEDRank and RepeatChoice cannot be distinguished.

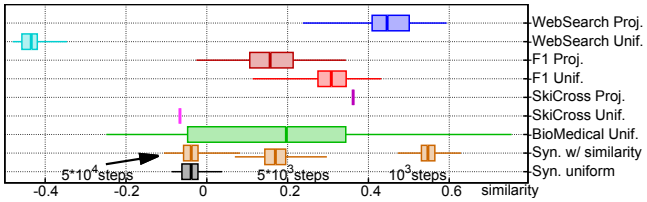

Figure 3: Distribution of the similarity for each group of datasets, grouped by type. Synthetic datasets with similarities are presented in three different configurations depending on the number of steps used in the Markov chain process generation.

to be preferred when using MEDRank.

While some algorithms show coherent behavior between synthetic and real datasets (cf. Table 4 and 5), others do not. Two observations highlight the need of a finer grained information on the datasets in order to understand the situation where algorithms perform well or do not.

First, when considering FaginDyn variants, FaginSmall performs better on 99.59% of the synthetic datasets. This observation is not repeated on real world datasets, as the two versions are even: FaginSmall performs better than FaginLarge in 49.52% of the datasets.

Second, BordaCount is observed on synthetic data to provide relatively good results and being positively influenced when the number of elements increases while this trend is not observed in real world datasets.

Considering datasets similarities could help understand algorithms behaviors. This will be explored in Section 7.2.

7.1.2 *Experimental computation time on uniformly generated datasets*

We now consider the time consumption for different values of n the number of elements in the datasets and fixed $m = 7$.

First of all, the average computing time needed by positional algorithms MEDRank, CopelandMethod, RepeatChoice and BordaCount to return a solution is very small: for datasets of $n = 400$ they take in average respectively $436\mu s$, $463\mu s$, $468\mu s$ and $574\mu s$. With the synthetic data, we observed that when n grows, MEDRank, CopelandMethod, BordaCount still return consensus of good quality compared to the other algorithms while being must faster. They are thus very good candidates for large datasets.

Second, considering only the two algorithms returning quality results, BioConsert strictly outperforms Ailon$\frac{3}{2}$ e.g. for $n = 40$ elements they take resp. $0.0083s$ and $50.373s$.

From Table 4, Table 5 and Figure 2, it appears that *BioConsert* is able to provide quality results in a very reasonable amount of time while positional approaches can provide answers very quickly but with lower quality.

7.2 Considering the similarity of datasets

Over the 10^5 datasets generated and used in this subsection, the exact solution is found in 99.91% of the datasets by the ExactAlgorithm with the computation time allocated. It ensures that no bias is introduced by datasets where it was not possible to compute the exact solution.

The Markov chain modeling designed in Section 6.1.2 converges to the uniform distribution when an infinite number of steps is considered. Knowing the number of steps needed to reach the uniform distribution with a relative error less than a given ε is out of the scope of this paper. Instead, we focus on two indicators to highlight that datasets generated with 50 000 steps are equivalent to uniformly generated datasets with the same numbers of elements and rankings: (1) the average similarity of uniformly generated datasets is $s = -0.0388$ while the Markov based datasets have a similarity of $s = -0.0384$ (note that with 50 steps $s = 0.88$, with 1000 steps $s = 0.55$ and with 5000 steps $s = 0.17$), (2) the results obtained on uniformly generated datasets with the same numbers of rankings and elements are equivalent to the results obtained when considering datasets generated with the Markov chain modeling.

Time consumption. Experiments conducted on synthetic datasets with different levels of similarities regarding the time consumption reveal two groups of algorithms: the algorithms from the first group take significant advantage of datasets with intrinsic similarities, while the second group is not impacted by the similarity. The local search algorithms are expected to be in the first group and they are: compared to not similar datasets (50 000 steps, $s = 0.04$), BioConsert proposes a consensus up to 57% faster with similar datasets (50 steps, $s = 0.88$). Similarly, ExactAlgorithm and Ailon$\frac{3}{2}$ also propose results faster on similar datasets: respectively 85% and 11% faster. The second group is made of BordaCount, CopelandMethod, FaginLarge, FaginSmall, KwikSort, Pick-a-Perm and RepeatChoice.

Considering the gap. Three interesting behaviors have been spotted (cf. Figure 4). First, KwikSort is positively influenced by the similarity, the average *gap* is 24 times smaller with very similar datasets (50 steps) compared to not similar datasets (50 000 steps). BioConsert has the same behavior: it always finds the optimal solution of similar datasets ($\leq$ 500 steps), and has an average *gap* = 0.02% with not similar datasets (50 000 steps). Observations on KwikSort are completed when using real-world dataset: KwikSort is indeed more efficient with similar datasets, but it is also negatively impacted by a negative similarity of datasets "WebSearch Unif" and "SkiCross Unif" (cf. Figure 3).

Second, BordaCount presents a very stable *gap* in Figure 4 for the different levels of similarity: the average *gap* varies from 3.6% to 4.0%. Surprisingly, this stability cannot be observed on real world datasets, where BordaCount is one of the worst algorithms on F1 unified datasets (30.2%) and is interesting on F1 projected datasets (3.7%).

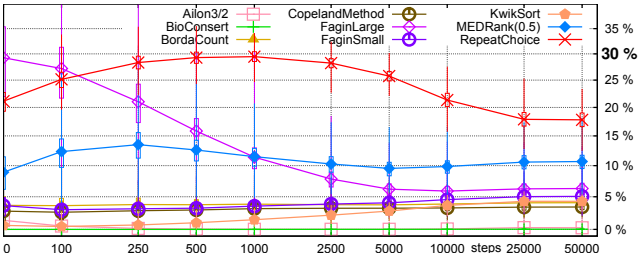

Figure 4: For synthetic datasets with similarities, gap plotted for different numbers of steps used during the generation (cf. Section 6.1.2).

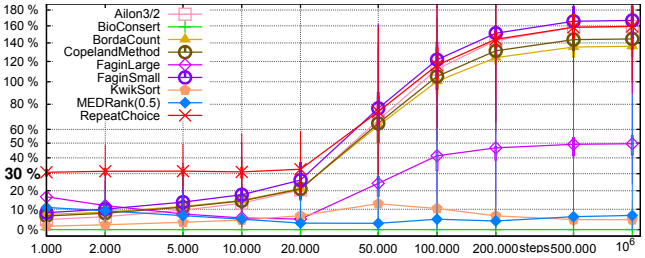

Figure 5: For unified synthetic datasets with similarities, gap plotted for different numbers of steps used during the generation (cf. Section 6.1.3).

Third, FaginLarge is negatively influenced by the similarity of the synthetic datasets: the average *gap* is more than 4.7 times larger with very similar datasets (50 steps) than with not similar datasets (50 000 steps). Again, this observation cannot be confirmed on real world datasets: while FaginSmall performs better than FaginLarge for WebSearch projected datasets which are similar, it is the opposite on F1 Unified datasets which are also similar.

Observations done on BordaCount or FaginDyn with synthetic datasets (cf. Figure 4) and which cannot be repeated on real world data (cf. Table 4 and Figure 3) highlight clearly that for some algorithms, knowing the size of a dataset and its similarity is still not enough to make an informed choice on whether using or not these algorithms. For this reason, the next section focuses on the influence of the standardization process.

Two approaches particularly benefit from the similarity in term of quality (Figure 4), namely BioConsert and KwikSort, while BioConsert and ExactAlgorithm benefit from similarity in terms of time consumption.

7.3 Similarity and Normalization

In Section 7.3.1 we compare two normalization processes, namely the unification and projection process. In Section 7.3.2 we study the influence of the unification process on the results produced by the algorithms.

7.3.1 *Unifying and projecting real datasets*

When considering a raw dataset with m rankings where the length of the longest ranking is l, it can be noticed that the projection process produces a dataset over 0 to l elements while the unification process produces a dataset over l to $l \times m$ elements.

The F1 datasets are seasons of F1 championship where each ranking is the order of arrival of pilots finishing the race. As done by [5], the projection removes 53.42%±25.03% of the pilots. Among the removed pilots we find the 1961 vice-champion and the 1970 champion. Those two pilots can clearly be qualified as relevant elements, not to be removed. In average, projected datasets are over 15.81 ± 8.53 elements while unified datasets are over 38.73 ± 11.39 elements.

When considering the WebSearch datasets, the projection removes in average $98.42\% \pm 0.89\%$ of the elements (procedure followed by [5]). Each ranking contains the top 1000 results for a search engine; the projection produces, in average, datasets over 40 ± 20 elements while unified datasets are over $2\,586 \pm 388$ elements (used in [3, 31]).

Using a process that takes into account the elements not present in all rankings appears to be a crucial need in real life use cases. Indeed the usually used projection process can remove relevant elements, such as champion in the F1 datasets. Nevertheless the increase of size between projected and unified datasets (there can be from 2.4 to 65 times more elements with unification) should be taken into account.

Having a large difference of size between projected and unified datasets is an indicator of the presence of large unifying buckets. As an example, in the WebSearch datasets, unification buckets have an average size of 1586 elements.

The next subsection studies these possibly large unification buckets in more detail.

7.3.2 *Unification impact on similar data*

The smallest the number of elements input rankings have in common before applying the unification process, the largest the size of unification buckets is expected to be. On unified synthetic datasets with similarities (cf. Section 6.1.3), the average size of buckets of similar datasets (generated with 10^3 steps) is 1.52 while it is 6.52 for not similar datasets (generated with 10^6 steps). Algorithms are expected to be cleaved into two categories depending on whether or not they consider the cost of untying elements.

We have on the one hand, BioConsert, KwikSort, and MEDRank which consider the importance of untying elements and are expected to be stable in quality, an assumption confirmed in this experimentation (cf. Figure 5).

On the other hand, BordaCount, CopelandMethod, and RepeatChoice are not able to consider the cost of untying elements. They are expected to be dramatically impacted by the unification process which can create a large ending tie. Experimentally they induce 15 times more disagreements with not similar and unified datasets than with similar unified datasets. The variation of quality of results for BordaCount and CopelandMethod with real world datasets which was not reproducible with synthetic datasets, whether they were similar or not, is now fully reproduced (cf. Figure 5 vs Figure 4).

FaginDyn variants had behaviors which were not consistent when only taking into account the size and similarity of the datasets (cf. Section 7.2). In this experiment, we clearly observe that the unification process, leading to the creation of a big ending bucket for datasets with no similarities, has a negative impact on FaginSmall. In the present experiment, favoring small buckets when there are large buckets in the input rankings is clearly a disadvantageous choice as shown in Figure 5.

On both unified synthetic and unified real world datasets CopelandMethod outperforms BordaCount, and they have the same performance on projected datasets.

The unification process as a standardization causes large ending buckets which are now understood as being the cause of bad performance of BordaCount and FaginSmall. Results obtained on real and synthetic datasets are consistent to each others now that the impact of the standardization process is understood.

7.4 Guidance based on known properties

Based on our experiments, we are now able to provide recommendations according to the known properties of the data to deal with, and the desired trade-off between time efficiency and expected quality of the results. Figure 6 illustrates the choice that could be proposed to the user, it has been generated with 100 uniformly generated datasets of $m = 7$ rankings over $n = 35$ elements. Results are representative of other uniformly generated datasets.

As a general outcome, BioConsert appears to be the best approach to follow in a very large number of cases. In extreme situations, some alternatives may be considered.

First, when highest quality results are mandatory and beside the ExactAlgorithm which provides optimal consensus, BioConsert should be favoured. While being reasonably more time consuming than other algorithms, it takes advantage of the similarity of datasets and it is independent of the standardization process applied.

Second, when extremely large datasets are considered (number of elements n>30 000), the implementation of BioConsert with a $\mathcal{O}(n^2)$ memory complexity can face physical limitation and may additionally take time to propose a consensus. KwikSort is then the best alternative for good quality results, and it is positively influenced by the similarity of datasets (cf. Figure 4).

If time is highly important then *BordaCount* is to be considered if only a few ties are involved whereas with large ties (possibly obtained by unification process) MEDRank is an excellent candidate.

8. CONCLUSION

We have conducted the first large-scale study of algorithms for rank aggregation with ties.

First, we have proposed an Integer Linear Programming algorithm to compute the optimal consensus ranking for the rank aggregation problem in the presence of ties. We have used the results of such exact solution to evaluate other approaches.

Second, we have reviewed the algorithms available in the literature and described which changes were needed in such algorithms to handle ties. We have considered a set of very diverse categories of algorithms which have all been reimplemented in a general framework and experimented both on available real-world and new synthetic datasets. Very importantly, we have been able to identify the data features to be considered to understand the behavior of algorithms (to then be able to guide users), namely, the size, level of similarity and normalization process. A total of 19 000 datasets of 10 to 500 elements are involved in our experiments and are all available at `http://rank-aggregation-with-ties.lri.fr/datasets/`.

We now discuss future work.

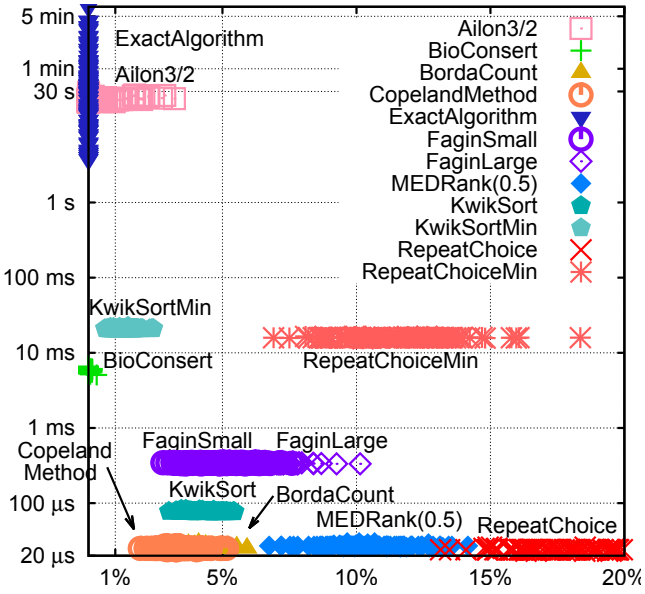

Figure 6: Computing time and gap achieved by algorithms for uniformly generated dataset of $m = 7$ rankings over $n = 35$ elements.

First, the surprising improvement shown by BordaCount and CopelandMethod when increasing the number of element for a fixed amount of ranking is being investigated under a theoretical point of view.

Second, unification and projection processes can be seen as two extreme variants of the same standardization process where the elements belonging to less than k rankings are removed, and the others are appended into a unification bucket when they are missing. Studying intermediate values of k would allow keeping a reasonable amount of data while ensuring the presence of relevant elements.

Third, considering strategies chaining several algorithms is another line of research to be explored (in the same spirit of [10]). In particular, simulated annealing techniques are known to produce high-quality consensus, but are time consuming. Chaining this kind of anytime approach to refine the solution produced by another (less time consuming) algorithm would allow to efficiently produce high-quality consensus.

9. ACKNOWLEDGEMENT

This work has partially been done at the Institute of Computational Biology. It has been supported in part by the CNRS PEPS FaSciDo "RankaBio".

Authors would like to thank Ulf Leser for very constructive feedback on this work and reviewers for their helpful comments on the paper.

10. REFERENCES

[1] N. Ailon. Aggregation of partial rankings, p-ratings and top-m lists. *Algorithmica*, 57(2):284–300, 2010.

[2] N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: ranking and clustering. *Journal of the ACM (JACM)*, 55(5):23, 2008.

[3] A. Ali and M. Meilă. Experiments with kemeny ranking: What works when? *Mathematical Social Sciences*, 64(1):28–40, 2012.

[4] J. P. Baskin and S. Krishnamurthi. Preference aggregation in group recommender systems for committee decision-making. In *Proceedings of the third ACM conference on Recommender systems*, pages 337–340. ACM, 2009.

[5] N. Betzler, R. Bredereck, and R. Niedermeier. Theoretical and empirical evaluation of data reduction for exact kemeny rank aggregation. *Autonomous Agents and Multi-Agent Systems*, pages 1–28, 2013.

[6] N. Betzler, J. Guo, C. Komusiewicz, and R. Niedermeier. Average parameterization and partial kernelization for computing medians. *Journal of Computer and System Sciences*, 77(4):774–789, 2011.

[7] T. Biedl, F. J. Brandenburg, and X. Deng. On the complexity of crossings in permutations. *Discrete Mathematics*, 309(7):1813–1823, 2009.

[8] J. Borda. Mémoire sur les élections au scrutin. *Histoire de l'academie royal des sciences*, pages 657 – 664, 1781.

[9] B. Brancotte and R. Milosz. Rank aggregation with ties is at least as difficult as rank aggregation without ties. Internal report, Univ. Paris-Sud - France.

[10] B. Brancotte, B. Rance, A. Denise, and S. Cohen-Boulakia. Conqur-bio: Consensus ranking with query reformulation for biological data. In *Data Integration in the Life Sciences*, volume 8574 of *LNCS*, pages 128–142. 2014.

[11] S. Chanas and P. Kobylański. A new heuristic algorithm solving the linear ordering problem. *Computational optimization and applications*, 6(2):191–205, 1996.

[12] S. Cohen-Boulakia, A. Denise, and S. Hamel. Using medians to generate consensus rankings for biological data. In *Scientific and Statistical Database Management*, volume 6809 of *LNCS*, pages 73–90. Springer, 2011.

[13] T. Coleman and A. Wirth. Ranking tournaments: Local search and a new algorithm. *Journal of Experimental Algorithmics (JEA)*, 14:6, 2009.

[14] V. Conitzer, A. Davenport, and J. Kalagnanam. Improved bounds for computing kemeny rankings. In *Proceedings of the 21st national conference on Artificial intelligence - Volume 1*, pages 620–626. AAAI Press, 2006.

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

[16] D. Coppersmith, L. Fleischer, and A. Rudra. Ordering by weighted number of wins gives a good ranking for weighted tournaments. In *Proc. of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 776–782, 2006.

[17] A. Davenport and J. Kalagnanam. A computational study of the kemeny rule for preference aggregation. In *Proc. of the 19th National Conference on Artifical Intelligence*, pages 697–702. AAAI Press, 2004.

[18] R. P. DeConde, S. Hawley, S. Falcon, N. Clegg, B. Knudsen, and R. Etzioni. Combining results of microarray experiments: a rank aggregation approach. *Statistical Applications in Genetics and Molecular Biology*, 5(1), 2006.

[19] P. Diaconis and R. L. Graham. Spearman's footrule as a measure of disarray. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 262–268, 1977.

[20] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *Proc. of the 10th World Wide Web conference*, pages 613–622. ACM, 2001.

[21] R. Fagin, R. Kumar, M. Mahdian, D. Sivakumar, and E. Vee. Comparing and aggregating rankings with ties. In *Proc. of the 23rd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 47–58. ACM, 2004.

[22] R. Fagin, R. Kumar, M. Mahdian, D. Sivakumar, and E. Vee. Comparing partial rankings. *SIAM Journal on Discrete Mathematics*, 20(3):628–648, 2006.

[23] R. Fagin, R. Kumar, and D. Sivakumar. Comparing top k lists. *SIAM Journal on Discrete Mathematics*, 17(1):134–160, 2003.

[24] R. Fagin, R. Kumar, and D. Sivakumar. Efficient similarity search and classification via rank aggregation. In *Proc. of the ACM SIGMOD int. conf. on Management of data*, pages 301–312. ACM, 2003.

[25] P. Flajolet, P. Zimmerman, and B. V. Cutsem. A calculus for the random generation of labelled combinatorial structures. *Theoretical Computer Science*, 132:1 – 35, 1994.

[26] F. Hivert and N. M. Thiéry. MuPAD-Combinat, an open-source package for research in algebraic combinatorics. *Sém. Lothar. Combin.*, 51:Art. B51z, 70 pp. (electronic), 2004.

[27] M. Jacob, B. Kimelfeld, and J. Stoyanovich. A system for management and analysis of preference data. *Proc. of the VLDB Endowment*, 7(12), 2014.

[28] J. G. Kemeny. Mathematics without numbers. *Daedalus*, 88(4):577–591, 1959.

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

[30] M. Meila, K. Phadnis, A. Patterson, and J. A. Bilmes. Consensus ranking under the exponential model. *arXiv preprint arXiv:1206.5265*, 2012.

[31] F. Schalekamp and A. van Zuylen. Rank aggregation: Together we're strong. In *ALENEX*, pages 38–51. SIAM, 2009.

[32] J. Sese and S. Morishita. Rank aggregation method for biological databases. *Genome Informatics Series*, pages 506–507, 2001.

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

[34] J. Stoyanovich, S. Amer-Yahia, S. B. Davidson, M. Jacob, T. Milo, et al. Understanding local structure in ranked datasets. In *CIDR*, 2013.

[35] A. Van Zuylen and D. P. Williamson. Deterministic pivoting algorithms for constrained ranking and clustering problems. *Mathematics of Operations Research*, 34(3):594–620, 2009.