Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

# Optimizing Positional Scoring Rules for Rank Aggregation

**Ioannis Caragiannis**
University of Patras
caragian@ceid.upatras.gr

**Xenophon Chatzigeorgiou**
University of Patras
chatzigeorgiou@ceid.upatras.gr

**George A. Krimpas**
University of Patras
krimpas@ceid.upatras.gr

**Alexandros A. Voudouris**
University of Patras
voudouris@ceid.upatras.gr

**Abstract**

Nowadays, several crowdsourcing projects exploit social choice methods for computing an aggregate ranking of alternatives given individual rankings provided by workers. Motivated by such systems, we consider a setting where each worker is asked to rank a fixed (small) number of alternatives and, then, a positional scoring rule is used to compute the aggregate ranking. Among the apparently infinite such rules, what is the best one to use? To answer this question, we assume that we have partial access to an underlying true ranking. Then, the important optimization problem to be solved is to compute the positional scoring rule whose outcome, when applied to the profile of individual rankings, is as close as possible to the part of the underlying true ranking we know. We study this fundamental problem from a theoretical point of view and present positive and negative complexity results. Furthermore, we complement our theoretical findings with experiments on real-world and synthetic data.

## 1 Introduction

Social choice theory (Brandt et al. 2016) studies voting rules (also known as social choice or social welfare functions) that compute a winning alternative or a ranking of the available alternatives from voter preferences. Typically, the preference of each voter is supposed to be a ranking over *all* available alternatives. We deviate from this assumption and, instead, we focus our attention to settings in which each voter (or, better, agent for our purposes) ranks only a small subset of the alternatives. Such *incomplete* rankings seem to be non-standard in the literature; (de Weerdt, Gerding, and Stein 2016; Dwork et al. 2001; Sculley 2007) are some notable exceptions.

The setting we have in mind is motivated by crowdsourcing (Law and von Ahn 2011) and rating applications. For example, assume that a requester would like to rank a huge set of alternatives using expert opinions from a crowd of workers. Asking each worker for her opinion on the whole set of alternatives (i.e., for a full ranking) would possibly result in poor information. Most probably, the worker will not be aware of most of the alternatives. Even if she tries to obtain additional information, coming up with consistent comparisons between alternatives that she knows well and alternatives that she has no idea about would be rather impossible, given their huge number. Instead, this task would be much easier if workers focused on small sets of alternatives. The requester could give each worker a different set of few alternatives to rank. Then, processing smaller inputs would be easier for the requester as well.

This approach has been recently exploited in the context of ordinal peer grading in MOOCs; see (Caragiannis, Krimpas, and Voudouris 2015; 2016; Raman and Joachims 2014; Shah et al. 2013) for approaches of this flavour. In such settings, the task of grading an exam with many participating students is outsourced to the students themselves. Each student is given a small number of exam papers to rank and the final grading (a ranking of all students) is obtained by aggregating the inputs provided by the students.

In a rating application we envision, users of a hotel booking system are asked to rank hotels they have stayed recently in a specific city and the rating application aims to compute a full ranking of the hotels (or, possibly, different rankings for different relevant criteria). Clearly, each user can provide meaningful feedback for just a few hotels. Again, in this scenario, the system might ask each user to focus only on a subset of the hotels she knows.

Besides the different sets of alternatives each individual is asked to rank in the above scenarios, another implicit feature is that there is an *underlying true ranking* of all alternatives (e.g., the ranking of exam papers in terms of their quality or the ranking of hotels in terms of their facilities) that we would like to compute when aggregating the individual preferences. Can we do so using simple voting-like rules? We follow an optimization approach which can be described with the following question: Assuming that we have partial knowledge of the underlying true ranking and access to sampled profiles, which is the rule that yields an outcome that is as consistent as possible to (our partial knowledge of) the underlying true ranking when applied to the sampled profiles?

We study the above question for *positional scoring rules* (or, simply, scoring rules), which have played a central role in social choice theory. Two factors that have led to this decision are their simplicity and effectiveness; simplicity follows by their definition and effectiveness is justified by our experimental results. In particular, we consider settings in which each agent is asked to rank the same number $d$ of

alternatives. A positional scoring rule in our setting is defined by a scoring vector $(s_1, s_2, ..., s_d)$. It takes as input the incomplete individual rankings of the agents and computes scores for alternatives as follows. An alternative gets $s_k$ points each time it is ranked $k$-th by an agent and its score is its total number of points. The final ranking is obtained by ordering all alternatives in terms of their scores, in non-increasing order. Now, given a profile of individual incomplete rankings and desired relations for pairs of alternatives (to be thought of as parts of the underlying true ranking) with corresponding weights indicating the importance of each relation, we would like to compute the positional scoring rule whose outcome, when applied on the profile, maximizes the total weight of the desired pairwise relations it satisfies. This is related to learning-theoretic studies where a scoring rule that is as consistent as possible to given examples is sought; e.g., see the paper by Boutilier et al. (2015) and Procaccia et al. (2009). The key difference of the current paper (besides our assumption on profiles with incomplete rankings) is its optimization flavour. We refer to this seemingly fundamental optimization problem as OptPSR.

Our technical contribution is as follows. We present an exact algorithm that solves OptPSR in time that depends exponentially only on the parameter $d$ (Section 3). Hence, our algorithm runs in polynomial time when $d$ is constant. For instances with high values of $d$, we show that a simple $t$-approval voting rule (that uses the scoring vector with $t$ 1s followed by $d - t$ 0s) yields a $1/d$-approximate solution. We show that this bound is tight by constructing an instance in which any approval voting rule is $1/d$-approximate. We prove that OptPSR is hard to approximate and present an explicit inapproximability bound of $23/24$. This result follows by an approximation-preserving reduction from the problem MAX-3LIN-2 of maximizing the number of satisfied equations in an over-determined system of linear equations modulo 2 and exploits a famous inapproximability result due to Håstad (2001). These results can be found in Section 4. In Section 5, we describe experiments on real-world and synthetic profiles. Our experimental results show that scoring rules perform remarkably well and recover almost $100\%$ of the desired constraints in many interesting scenarios; this justifies our choice to study scoring rules (and the optimization problem OptPSR) in the first place.

We begin with preliminary definitions in Section 2 and conclude with open problems in Section 6. Due to lack of space, the full proof of our inapproximability result is omitted.

## 2 Problem statement

We consider settings with a set of *agents* $N$ and a set of *alternatives* $A$. Agent $i$ expresses her preference over a subset $A_i \subseteq A$ of alternatives; her preference is a (strict) ranking of the alternatives in $A_i$. A preference profile (or simply, a *profile*) consists of the preferences of all agents. In this work, we assume that all agents have the *same* number $d \geq 2$ of alternatives in their preference, i.e., $|A_i| = d$ for each agent $i$. However, different agents may have different alternatives in their preferences.

A social welfare function takes as input a profile $\Pi$ and it outputs a ranking of all alternatives in $A$. A positional scoring rule (or, simply, a scoring rule) is a social welfare function that uses a scoring vector $\mathbf{s} = (s_1, ..., s_d)$ with $s_i \geq s_{i+1}$ for $i = 1, ..., d-1$ and $s_d \geq 0$; the alternative at position $k$ in each vote is assigned $s_k$ points and the ranking of the alternatives is produced by ordering them in monotone non-increasing order in terms of their total points (or score). Formally, for an alternative $x$, let $\nu_j(x, \Pi)$ denote the number of agents that rank $x$ at position $j$ in profile $\Pi$. Then, given a scoring rule $\mathbf{s}$, the score of alternative $x$ is defined as

$$\mathtt{sc}_{\mathbf{s}}(x, \Pi) = \sum_{j=1}^{d} \nu_j(x, \Pi) \cdot s_j.$$

We also assume that we have access to a set of *constraints* $C$ that represents our (possibly partial) knowledge to an objective set of pairwise relations between the alternatives. Each constraint in $C$ is given by an ordered pair of alternatives $(x, y)$, has a corresponding non-negative weight (of importance) $w(x, y)$, and requires that alternative $x$ is ranked higher than alternative $y$ in the outcome of the scoring rule $\mathbf{s}$. For a pair of alternatives $(x, y)$, let $\delta_j(x, y, \Pi) = \nu_j(x, \Pi) - \nu_j(y, \Pi)$. Now, observe that, in order for alternative $x$ to be ranked above $y$ with certainty in the final ranking, it must be $\mathtt{sc}_{\mathbf{s}}(x, \Pi) > \mathtt{sc}_{\mathbf{s}}(y, \Pi)$ and, equivalently,

$$\sum_{j=1}^{d} \delta_j(x, y, \Pi) \cdot s_j > 0.$$

Using $\boldsymbol{\delta}(x, y, \Pi) = (\delta_1(x, y, \Pi), ..., \delta_d(x, y, \Pi))$, the above expression can be compactly written as the dot product $\boldsymbol{\delta}(x, y, \Pi) \cdot \mathbf{s} > 0$.

For our purposes, instead of thinking of a profile $\Pi$ as the set of rankings provided by the agents, it is convenient to describe it using the quantities $\boldsymbol{\delta}(x, y, \Pi)$ for every constraint $(x, y)$ in $C$; we use the notation $\boldsymbol{\delta}(\Pi)$ to denote the set of these quantities and will simply refer to it as the profile.

Now, problem OptPSR (standing for "optimizing positional scoring rules") is defined as follows. We are given a profile $\boldsymbol{\delta}(\Pi)$ and a set $C$ of constraints. The goal of OptPSR is to find the scoring rule $\mathbf{s}$ that produces a ranking of all alternatives so that the total weight (or gain)

$$\mathtt{g}(\mathbf{s}, \boldsymbol{\delta}(\Pi), C) = \sum_{(x,y) \in C} w(x, y) \cdot \mathbb{I}\left\{\boldsymbol{\delta}(x, y, \Pi) \cdot \mathbf{s} > 0\right\},$$

of satisfied constraints is maximized. The quantity $\mathbb{I}\{X\}$ takes value 1 if $X$ is true and 0 otherwise.

Let us now give an equivalent view of OptPSR. A scoring rule $\mathbf{s}$ can be thought of as a point in $\mathbb{R}^d$, and, in particular, in the region $R_0$ of $\mathbb{R}^d$ formed by the inequalities $s_i - s_{i+1} \geq 0$ for $i = 1, ..., d-1$ and $s_d \geq 0$ that define all valid scoring vectors. We can define subregions of $R_0$ by considering any subset $C' \subseteq C$ of constraints and the inequality $\boldsymbol{\delta}(x, y, \Pi) \cdot \mathbf{s} > 0$ for every constraint associated with the pair of alternatives $(x, y) \in C'$ and the inequality $\boldsymbol{\delta}(x, y, \Pi) \cdot \mathbf{s} \leq 0$ for every constraint $(x, y) \in C \setminus C'$. In this

way, the collection of all subsets of constraints in $C$ partition $R_0$ into disjoint subregions (of course, some of them may be infeasible). Hence, in order to maximize $\mathtt{g}(\mathbf{s}, \boldsymbol{\delta}(\Pi), C)$, it suffices to find any point $\mathbf{s}$ in the non-empty subregion of $R_0$ that satisfies the subset of constraints with maximum total weight.

To do so, we can enumerate all subsets of constraints of $C$, check feasibility of the corresponding regions using linear programming, and report any point in the subregion that yields the highest gain. This algorithm takes time polynomial in $2^{|C|}$ and $d$, assuming that it receives $\boldsymbol{\delta}(\Pi)$ and $C$ as input. In the next section, we will present an algorithm that uses a more clever enumeration of the feasible subregions in order to get the one that yields the maximum gain.

## 3 An improved OptPSR algorithm

We will present another (exact) OptPSR algorithm whose running time depends exponentially only on the parameter $d$ and, hence, is polynomial when $d$ is a constant.

The algorithm computes a pool of non-empty subregions of $R_0$, each of which satisfies a different subset of constraints. Initially, the pool consists of region $R_0$ only and is updated as new constraints of $C$ are considered. When a new constraint is considered, each region in the current pool can be split into two subregions consisting of the points that satisfy the constraint and those that do not satisfy it, respectively; the whole region is retained in the pool if all its points satisfy or (exclusively) do not satisfy the constraint.

In particular, the algorithm considers the constraints of $C$ one by one. At each step $t$ of the algorithm, a pool $\mathcal{P}$ of regions is kept; at the beginning of each step, all regions in the pool are active. For each region $R$ in $\mathcal{P}$, the algorithm keeps the gain $\mathtt{val}(R)$ that is obtained by the constraints which have been considered until step $t$ and are satisfied by scoring vectors of region $R$. The algorithm begins its execution having only region $R_0$ in the pool. When a new constraint $(x, y)$ with weight $w(x, y)$ is considered, the algorithm attempts to update each active region $R$ of $\mathcal{P}$ as follows. It defines the candidate regions $R^{xy}$ and $R^{\neg xy}$ such that

- $R^{xy}$ is defined by the inequalities that form $R$, together with inequality $\boldsymbol{\delta}^{xy} \cdot \mathbf{s} > 0$ (that defines the set of points that satisfy constraint $(x, y)$), and
- $R^{\neg xy}$ is defined by the inequalities that form $R$, together with inequality $\boldsymbol{\delta}^{xy} \cdot \mathbf{s} \leq 0$ (that defines the set of points that do not satisfy constraint $(x, y)$).

If both $R^{xy}$ and $R^{\neg xy}$ are non-empty (i.e., the corresponding sets of inequalities are feasible), the algorithm includes both $R^{xy}$ and $R^{\neg xy}$ in $\mathcal{P}$ as inactive, sets their gains $\mathtt{val}(R^{xy}) := \mathtt{val}(R) + w(x, y)$ and $\mathtt{val}(R^{\neg xy}) := \mathtt{val}(R)$, and removes region $R$ from the pool. If only $R^{xy}$ is feasible (and $R^{\neg xy}$ is infeasible), $\mathtt{val}(R)$ is increased by $w(x, y)$. If only $R^{\neg xy}$ is feasible, the algorithm does nothing. In the last two cases, no new region is added to the pool. Clearly, it cannot be the case that both $R^{xy}$ and $R^{\neg xy}$ are infeasible. Note that feasibility can be checked efficiently by solving linear programs with $d$ variables and up to $|C|$ constraints. At the end of step $t$ (i.e., when there is no other active region in the pool to be considered), the inactive regions become active and the algorithm proceeds with step $t + 1$.

When all constraints of $C$ have been considered, the algorithm computes the active region $R^*$ with maximum

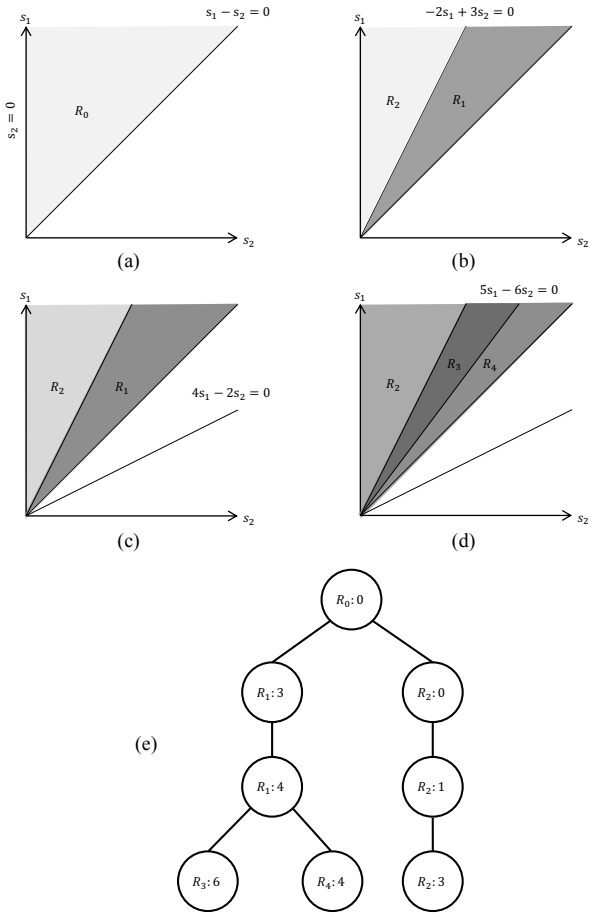


Figure 1: An example of how the algorithm works on a simple instance $\Pi$ with $d = 2$. The set $C$ has three constraints $(x_1, y_1)$, $(x_2, y_2)$, and $(x_3, y_3)$ with corresponding weights 3, 1 and 2. The profile is such that $\delta(x_1, y_1, \Pi) = (-2, 3)$, $\delta(x_2, y_2, \Pi) = (4, -2)$, and $\delta(x_3, y_3, \Pi) = (5, -6)$. So, the constraints define the inequalities $-2s_1 + 3s_2 > 0$, $4s_1 - 2s_2 > 0$, and $5s_1 - 6s_2 > 0$. (a) Initially, the algorithm has region $R_0$ in the pool. (b) At the next step, the algorithm considers constraint $(x_1, y_1)$ and replaces $R_0$ with regions $R_1$ (which is the subregion of $R_0$ with $-2s_1 + 3s_2 > 0$ that satisfies the first constraint and has gain 3) and $R_2$ (with gain 0). (c) Next, constraint $(x_2, y_2)$ leaves both regions $R_2$ and $R_3$ in the pool. (d) Finally, the third constraint $(x_3, y_3)$ replaces region $R_1$ by regions $R_3$ and $R_4$. In (e), the evolution of the content of the pool, together with the gains of the corresponding subregions are illustrated. The region with the maximum gain is $R_3$ and the algorithm will output some scoring vector from this region.

$\texttt{val}(R^*)$ and returns any scoring vector in $R^*$. An example of an execution of the algorithm with $d = 2$ is depicted in Figure 1.

**Theorem 1.** *Given an instance of* OptPSR *consisting of a set of constraints $C$ and a profile $\boldsymbol{\delta}(\Pi)$, the algorithm above correctly returns a solution in time $\mathcal{O}(|C|^d \cdot \text{poly}(|C|, d))$.*

*Proof.* The correctness of the algorithm should be apparent. It considers the whole space of points in $\mathbb{R}^d$ which corresponds to scoring vectors and divides it into all (sub)regions defined for every inclusion-maximal subset of constraints that are satisfied simultaneously. Among all these regions, it finds the one with points that correspond to scoring vectors that satisfy constraints of $C$ with maximum total weight.

Expanding $R_0$ into the regions in the pool when the last constraint of $C$ is considered can be thought of as a non-complete binary tree $T$ with nodes corresponding to regions (see Figure 1 for an example). $T$ is rooted at a node corresponding to $R_0$ and is such that each node at level $t-1$, corresponding to a region $R$, has two children at level $t$ if the region $R$ was split in and replaced by two subregions at step $t$ and has one child otherwise (indicating that the region was retained in the pool during step $t$). The total time required to find all regions is proportional to the size of $T$. Since all non-leaf nodes have at least one child, the size of $T$ is at most its height $|C|$ times the number of leaves. The number of leaves is essentially the number of different non-empty regions, which is upper-bounded by the number of different sign patterns that the quantities $\boldsymbol{\delta}^{xy} \cdot \mathbf{s}$ define for each constraint $(x, y)$ in $C$. Since these $|C|$ quantities are linear functions over the $d$ coordinates of vector $\mathbf{s}$, a result due to Alon (1996) (see also Warren (1968)) yields that the total number of different sign patterns is at most $\left(\frac{8e|C|}{d}\right)^d$. For each of the nodes of $T$, feasibility can be checked by solving two linear programs with $d$ variables and at most $|C|$ constraints in time $\text{poly}(|C|, d)$. The theorem follows. □

By Theorem 1, we obtain the following corollary. For comparison, the naive algorithm presented at the end of the previous section is polynomial in the very special case where $|C|$ is at most logarithmic in $d$.

**Corollary 2.** *The algorithm solves instances of* OptPSR *with constant $d$ in polynomial time.*

## 4 Approximating OptPSR

As the running time of the exact algorithm of the previous section depends exponentially on $d$, our aim here is to design much faster (i.e., polynomial-time) algorithms that compute approximate OptPSR solutions. As we will see, an extremely simple scoring rule achieves a $1/d$-approximation, i.e., the total weight of the constraints it satisfies is at least $1/d$ times the total weight satisfied by an optimal OptPSR solution. For $t \in [d]$, the $t$-approval rule is a positional scoring rule that uses the scoring vector that has 1 in the first $t$ positions and 0 in the remaining ones.

**Theorem 3.** *For every instance of* OptPSR *with parameter $d$, there exists some $t \in [d]$ so that $t$-approval is a $1/d$-approximate solution. This bound is tight.*

*Proof.* For the lower bound, consider a profile $\boldsymbol{\delta}(\Pi)$ and set of constraints $C$. We partition the constraints of $C$ into $d$ disjoint sets $C_1, C_2, ..., C_d$ so that the $t$-th set is defined as

$$C_t = \left\{ (x, y) \in C : \sum_{k=1}^{\ell} \delta_k(x, y, \Pi) \leq 0 \text{ for all } \ell \leq t-1 \text{ and } \sum_{k=1}^{t} \delta_k(x, y, \Pi) > 0 \right\},$$

for $t = 1, 2, ..., d$. Observe that the $t$-approval rule satisfies all constraints in the set $C_t$. Actually, set $C_t$ is defined as the set of constraints that are satisfied by $t$-approval but not by $\ell$-approval for $\ell < t$. Hence, the sets $C_1, ..., C_d$ are disjoint and there exists $t^* \in [d]$ (with $\mathbf{t}^*$ being the scoring vector of $t^*$-approval) such that $\mathfrak{g}(\mathbf{t}^*, \boldsymbol{\delta}(\Pi), C) \geq \sum_{(x,y)\in C_{t^*}} w(x, y) \geq \frac{1}{d} \sum_{(x,y)\in C} w(x, y)$. As the maximum possible gain cannot exceed $\sum_{(x,y)\in C} w(x, y)$, we have that $t^*$-approval is $1/d$-approximate as desired.

For the upper bound, we will present an OptPSR instance such that any $t$-approval, with $t \in [d]$, is (at most) $1/d$-approximate. The instance has $d$ pairs of alternatives $(x_t, y_t)$ as constraints with $w(x_t, y_t) = 1$ for $t \in [d]$. We will build a profile $\Pi^*$ so that the $t$-approval rule satisfies only constraint $(x_t, y_t)$, while there exists a scoring rule that simultaneously satisfies all constraints. Consider quantities $a_1, a_2, ..., a_d$ with positive integer values such that $\sum_{j=1}^{d} \frac{1}{1+a_j} < 1$. The profile is defined as follows:

- Alternative $x_1$ appears $a_1$ times in position 1, and alternative $y_1$ appears $1 + a_1$ times in position 2. This means that $\delta_1(x_1, y_1, \Pi^*) = a_1$, $\delta_2(x_1, y_1, \Pi^*) = -1 - a_1$ and $\delta_j(x_1, y_1, \Pi^*) = 0$ for $j \geq 3$.
- For $2 \leq t \leq d-1$, alternative $x_t$ appears $1 + a_t$ times in position $t$, and alternative $y_t$ appears once in position 1 and $1 + a_t$ times in position $t + 1$. This means that $\delta_1(x_t, y_t, \Pi^*) = -1$, $\delta_t(x_t, y_t, \Pi^*) = 1 + a_t$, $\delta_{t+1}(x_t, y_t, \Pi^*) = -1 - a_t$ and $\delta_j(x_t, y_t, \Pi^*) = 0$ for $j \notin \{1, t, t+1\}$.
- Alternative $x_d$ appears $1 + a_d$ times in position $d$, and alternative $y_d$ appears once in position 1. This means that $\delta_1(x_d, y_d, \Pi^*) = -1$, $\delta_d(x_d, y_d, \Pi^*) = 1 + a_d$ and $\delta_j(x_d, y_d, \Pi^*) = 0$ for $2 \leq j \leq d-1$.
- The rest of the positions in the votes are filled with additional alternatives that do not appear in the constraints.

Observe that, for $t \in [d]$, we have that $\sum_{j=1}^{t} \delta_j(x_t, y_t, \Pi^*) = a_t > 0$, $\sum_{j=1}^{t-1} \delta_j(x_t, y_t, \Pi^*) = -1$ and $\sum_{j=1}^{\ell} \delta_j(x_t, y_t, \Pi^*) = -1$ for $\ell > t$. Hence, the $t$-approval rule satisfies only constraint $t$ for a total weight of 1.

Now we will show that there exists a scoring rule $\mathbf{s} = (s_1, s_2, ..., s_d)$ that satisfies all constraints. Let $\epsilon > 0$ be some arbitrary small constant and consider the scoring vector $\mathbf{s}$ with $s_1 = \frac{\epsilon \sum_{j=1}^{d} \frac{1}{1+a_j}}{1-\sum_{j=1}^{d} \frac{1}{1+a_j}}$, $s_2 = \frac{a_1 s_1 - \epsilon}{1+a_1}$, and $s_i = s_{i-1} - \frac{s_1+\epsilon}{1+a_{i-1}}$ for $i \geq 3$. First, observe that this is a valid

scoring rule since, it is clear that $s_i \geq s_{i+1}$ for all $i \in [d-1]$ and, furthermore, it can be easily seen that $s_d \geq 0$ as well. Moreover, this scoring rule satisfies all constraints since $\sum_{j=1}^{d} \delta_j(x_t, y_t, \Pi^*) \cdot s_j = \epsilon$ for every $t \in [d]$. Hence, any $t$-approval is $1/d$-approximate. □

On the negative side, we show that our problem is not only hard, but also hard to approximate within some constant.

**Theorem 4.** *For every constant $\eta > 0$,* OptPSR *is hard to approximate within $23/24 + \eta$.*

Due to lack of space, we just sketch the proof of Theorem 4 here. We use a reduction from MAX-3LIN-2, the problem of maximizing the number of satisfied equations in an over-determined system of linear equations modulo 2. An instance of MAX-3LIN-2 consists of $n$ binary variables $x_i \in \{0, 1\}$ and $m$ equations of the forms $x_i \oplus x_j \oplus x_k = 0$ and $x_i \oplus x_j \oplus x_k = 1$, where $\oplus$ denotes addition modulo 2 and its objective is to find an assignment to the variables so that the number of satisfied equations is maximized.

Given an instance of MAX-3LIN-2, our reduction constructs in polynomial-time an instance of OptPSR that has a scoring rule that satisfies constraints of total weight $11m+L$ if and only if the MAX-3LIN-2 instance has an assignment satisfying $L$ equations. A famous result by Håstad (2001) states that it is hard to distinguish in time polynomial in $n$ and $m$ whether a given instance of MAX-3LIN-2 has an assignment that satisfies at least $(1 - \eta')m$ equations or any assignment satisfies at most $(1/2 + \eta')m$ equations, for any constant $\eta' > 0$. As a consequence of our reduction, we obtain that it is hard to distinguish between instances of OptPSR that have a scoring rule that satisfies constraints of total weight at least $(12 - \eta')m$ and instances of OptPSR in which the total weight of the constraints satisfied by any scoring rule is at most $(23/2 + \eta')m$. An inapproximability bound of $23/24 + \eta$ (for every constant $\eta > 0$) then follows by standard arguments.

Without loss of generality, we can assume that the scoring vectors $\mathbf{s} = (s_1, s_2, ..., s_d)$, that we seek for, have $s_1 = d$ and the remaining scores are defined in terms of $d - 1$ variables $a_1, a_2, ...a_{d-1} \geq 0$ as $s_{i+1} = s_i - a_i$ for $i = 1, ..., d-1$ so that $\sum_{i=1}^{d-1} a_i \leq d$. Hence, a constraint can be expressed as a linear inequality of the variables $a_j$ with $j \in [d-1]$. The assumption that $s_1 = d$ allows for inequalities that have non-zero constant terms. We only define linear inequalities corresponding to constraints here; in the full proof, we also construct the profile and specify the constraints as pairs of alternatives and corresponding weights that are consistent to these linear inequalities. Let $m_i$ be the number of equations in which variable $x_i$ participates. The instance of OptPSR is then as follows:

- For every variable $x_i$, we have the four inequalities $a_i > 0$, $a_i < \epsilon$, $a_i > 1$ and $a_i < 1 + \epsilon$ of weight $m_i$ each.
- For every equation, there are four inequalities of unit weight each:
  - if the equation is of the form $x_i \oplus x_j \oplus x_k = 0$, the inequalities are $a_i + a_j + a_k > 0$, $a_i + a_j + a_k < \epsilon$, $a_i + a_j + a_k > 2$ and $a_i + a_j + a_k < 2 + \epsilon$, and
  - if the equation is of the form $x_i \oplus x_j \oplus x_k = 1$, the inequalities are $a_i + a_j + a_k > 1$, $a_i + a_j + a_k < 1 + \epsilon$, $a_i + a_j + a_k > 3$ and $a_i + a_j + a_k < 3 + \epsilon$.

The parameter $\epsilon$ is (polynomially) small but strictly positive.

The crucial observation is that for every quadruple of inequalities corresponding to a variable or an equation, any value of the variables involved satisfy at least two and at most three of them. For example, values of $a_i$ in $(0, \epsilon)$ or $(1, 1 + \epsilon)$ satisfy three among the inequalities corresponding to variable $x_i$, while any other value satisfies exactly two of them. In a sense, these two ranges of values simulate the assignment of values 0 and 1 to variable $x_i$ in the MAX-3LIN-2 instance. The weight $m_i$ for these inequalities guarantees that variable $a_i$ will never take any value outside these intervals. Similarly, the quadruple of inequalities for an equation aims to control the fact that the equation is satisfied or not.

## 5 Experiments

We have conducted experiments for two different scenarios; we refer to them as *ppl* and *col*. In these two scenarios, we used as alternatives 48 countries and 36 cities, respectively. In both cases, the alternatives were used to define 392 different sets consisting of six alternatives each. The alternatives have been distributed to the different sets almost uniformly; each country appears in at least 47 and at most 52 sets and each city appears in at least 57 and at most 70 sets.

We used both real-world and synthetic data. Real-world data were collected as input from 392 participants in a technology exhibition at our home institution. Each of them was given two distinct sets of six countries and six cities. They were asked to rank the countries in terms of their population and the cities in terms of their cost of living. Synthetic data were obtained by simulating 392 agents who use the Plackett-Luce and Bradley-Terry noise models in order to compute random rankings.

The Bradley-Terry model (Bradley and Terry 1952) (BT, in short) is used by an agent in order to decide relations between all pairs of alternatives in her set as follows. Consider a pair of alternatives $(x, y)$ with corresponding utilities (populations or cost of living indices) $u_x$ and $u_y$. The agent decides to rank $x$ above $y$ with probability $\frac{u_x}{u_x+u_y}$ and $y$ above $x$ with probability $\frac{u_y}{u_x+u_y}$. If the relative ranks of all pairs of alternatives (that have been computed separately) do not define a ranking, the whole process is repeated.

In the Plackett-Luce model (Luce 1959; Plackett 1975) (PL, in short), an agent decides the ranking of the alternatives in her set sequentially. Let $B$ be the set of alternatives the agent has to rank. Starting from the first position, the next undetermined position in the ranking is filled by alternative $x \in B$ with probability $\frac{u_x}{\sum_{y \in B} u_y}$. After a random selection, the chosen alternative is removed from $B$ and the process continues for the next undetermined position and the remaining alternatives until all positions are filled.

The set of constraints were defined using population data for the 48 countries from `en.wikipedia.org` and cost of living index data from `numbeo.com`. In particular, in the ppl scenario, we have a constraint for each pair of coun-

tries $x$ and $y$ so that $x$ is more populous than $y$. We consider two different weightings for constraints using weight that is either 1 or equal to the population difference between countries $x$ and $y$. Unit weights are used when we care only about maximizing the number of correctly recovered population comparison between countries. However, there might be pairs that are really important to recover correctly, while some others are not that important. For example, it is important to conclude that China is ranked above Switzerland (their population difference is 1.3 billions) but an error in the comparison between Cuba and Belgium (both with populations around 11 millions) would not be that severe. Analogously, in the col scenario, we have a constraint for every pair of cities $x$ and $y$ so that $x$ has higher cost of living index than $y$. The weight of the corresponding constraint is either 1 or equal to the cost of living index difference between the two cities.

Since all the profiles we experimented with have $d = 6$, one would expect that the exact algorithm presented in Section 3 would be the obvious choice in order to come up with the optimal scoring rule. Unfortunately, for the size of OptPSR instances we considered (with 1128 constraints for ppl and 630 constraints for col), this algorithm turned out to be really slow, even after implementing several heuristics that yield minor improvements to performance. This rather disappointing outcome, together with the fact that $d$ is small, forced us to consider scoring vectors with discretized scores (e.g., which are multiples of 0.05 or 0.02) in order to come up with approximations of the optimal scoring rule. This approach has yielded the vectors $(1, 0.5, 0.35, 0.2, 0.15, 0.05)$ and $(1, 0.65, 0.65, 0.35, 0.3, 0.25)$ for the ppl profile with unweighted and weighted constraints and the vectors $(1, 0.9, 0.3, 0.3, 0.24, 0)$ and $(1, 0.68, 0.68, 0.5, 0.22, 0.22)$ for the col profile with unweighted and weighted constraints, respectively.

We compare the optimal OptPSR solution (obtained in this way) to several well-known scoring rules such as the Borda count (defined by the scoring vector $(d, d-1, ..., 1)$), the harmonic rule (also known as Dowdall; defined by the scoring vector $(1, 1/2, ..., 1/d)$), and $t$-approval rules. Tables 1 and 2 show the performance of these scoring rules in all OptPSR instances we experimented with.

In Table 1, which contains data for instances with unweighted constraints, we observe that Borda and Harmonic outperform all approval rules in all cases besides the col scenario with PL agents, where 4-approval is slightly better than Harmonic. Also, there are cases (e.g., ppl profile with BT agents) where Borda performs better than Harmonic and vice versa (e.g., see the results for real-world data). In all scenarios, the values of Borda, Harmonic and the best approval rule have an average difference of 2.3%, 2.36% and 3.1%, respectively, from the optimal values (with maximum difference values of 3.43%, 5.07% and 4.32% that are all observed for the col profile with PL agents). Interestingly, approval rules have amazingly better performance than what their worst-case analysis from Theorem 3 indicates.

Clearly, Table 2 shows significantly better results from (almost) all scoring rules on OptPSR instances with weighted constraints. This is to be expected since solutions improve significantly when heavy pairwise relations are correctly recovered. Here, Borda, Harmonic and the best approval are closer to the optimal performance. Now, the average distance is 1.05%, 1.08%, and 1.2%; again, the largest differences are observed for the col profile with PL agents.

| | real data | | synthetic (BT) | | synthetic (PL) | |
|---|---|---|---|---|---|---|
| rule | ppl | col | ppl | col | ppl | col |
| opt. | 81.83 | 83.97 | 94.54 | 93.74 | 93.19 | 88.20 |
| borda | 79.87 | 81.43 | 93.16 | 91.03 | 91.59 | 84.67 |
| harm. | 80.94 | 82.54 | 92.84 | 91.34 | 90.50 | 83.13 |
| 1-app. | 77.75 | 78.09 | 83.69 | 87.89 | 83.90 | 75.72 |
| 2-app. | 78.19 | 79.36 | 89.71 | 90.65 | 88.72 | 81.29 |
| 3-app. | 79.43 | 80.48 | 91.69 | 90.93 | 90.30 | 83.73 |
| 4-app. | 77.57 | 79.68 | 90.00 | 89.44 | 89.70 | 83.88 |
| 5-app. | 73.14 | 72.86 | 81.08 | 84.45 | 82.83 | 80.66 |
| 6-app. | 32.53 | 51.43 | 32.54 | 51.43 | 32.54 | 51.43 |

Table 1: Performance (as percentage of the total weight of all constraints) of scoring rules on instances with unweighted constraints. For synthetic profiles with BT and PL agents, the simulation was repeated 500 times; the values shown here are averages.

| | real data | | synthetic (BT) | | synthetic (PL) | |
|---|---|---|---|---|---|---|
| rule | ppl | col | ppl | col | ppl | col |
| opt. | 95.98 | 92.93 | 99.66 | 98.69 | 99.48 | 96.02 |
| borda | 94.56 | 91.57 | 99.49 | 97.66 | 99.23 | 93.96 |
| harm. | 95.42 | 92.01 | 99.42 | 97.80 | 99.02 | 92.58 |
| 1-app. | 94.85 | 89.84 | 98.59 | 96.48 | 97.86 | 86.34 |
| 2-app. | 95.24 | 90.40 | 99.29 | 97.67 | 98.86 | 91.50 |
| 3-app. | 93.68 | 90.83 | 99.05 | 97.70 | 99.04 | 93.37 |
| 4-app. | 92.63 | 90.06 | 97.46 | 96.91 | 98.38 | 93.51 |
| 5-app. | 84.61 | 82.00 | 87.94 | 93.81 | 91.69 | 90.99 |
| 6-app. | 39.66 | 57.04 | 39.88 | 56.96 | 39.88 | 56.96 |

Table 2: Performance of scoring rules on instances with weighted constraints. Again, for synthetic profiles with BT and PL agents, the values indicate average performance from 500 simulations.

## 6 Open problems

Our work reveals several open problems. First, we would like to determine the approximability of OptPSR. Is there a polynomial time algorithm with constant approximation ratio? Second, we would like to design an exact algorithm that is practical. Our ambitious goal is to be able to solve OptPSR instances like the ones we considered in our experiments. Third, we would like to analyze theoretically scoring rules in random profiles that have been produced by Plackett-Luce or Bradley-Terry agents. Also, considering agents following other noise models that are close to real-world agents definitely deserves investigation.

## 7 Acknowledgments

This work was partially supported by the Caratheodory research grant E.114 from the University of Patras and by a Ph.D. Scholarship from the Onassis Foundation.

## References


Alon, N. 1996. Tools from higher algebra. In Graham, R. L.; Grötschel, M.; and Lovász, L., eds., *Handbook of Combinatorics*, volume 2. MIT Press. 1749–1783.

Boutilier, C.; Caragiannis, I.; Haber, S.; Lu, T.; Procaccia, A. D.; and Sheffet, O. 2015. Optimal social choice functions: A utilitarian view. *Artificial Intelligence* 227:190–213.

Bradley, R. A., and Terry, M. E. 1952. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika* 39(3/4):324–345.

Brandt, F.; Conitzer, V.; Endriss, U.; Lang, J.; Moulin, H.; and Procaccia, A. D. 2016. *Handbook of Computational Social Choice*. Cambridge University Press.

Caragiannis, I.; Krimpas, G. A.; and Voudouris, A. A. 2015. Aggregating partial rankings with applications to peer grading in massive online open courses. In *Proceedings of the 14th International Conference on Autonomous Agents & Multiagent Systems (AAMAS)*, 675–683.

Caragiannis, I.; Krimpas, G. A.; and Voudouris, A. A. 2016. How effective can simple ordinal peer grading be? In *Proceedings of the 17th ACM Conference on Economics and Computation (EC)*, 323–340.

de Weerdt, M. M.; Gerding, E. H.; and Stein, S. 2016. Minimising the rank aggregation error. In *Proceedings of the 15th International Conference on Autonomous Agents & Multiagent Systems (AAMAS)*, 1375–1376.

Dwork, C.; Kumar, R.; Naor, M.; and Sivakumar, D. 2001. Rank aggregation methods for the web. In *Proceedings of the 10th International World Wide Web Conference (WWW)*, 613–622.

Håstad, J. 2001. Some optimal inapproximability results. *Journal of the ACM* 48(4):798–859.

Law, E., and von Ahn, L. 2011. *Human computation*. Morgan & Claypool Publishers.

Luce, R. D. 1959. *Individual choice behavior: A theoretical analysis*. Wiley.

Plackett, R. L. 1975. The analysis of permutations. *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 24(2):193–202.

Procaccia, A. D.; Zohar, A.; Peleg, Y.; and Rosenschein, J. S. 2009. The learnability of voting rules. *Artificial Intelligence* 173(12-13):1133–1149.

Raman, K., and Joachims, T. 2014. Methods for ordinal peer grading. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 1037–1046.

Sculley, D. 2007. Rank aggregation for similar items. In *Proceedings of the 7th SIAM International Conference on Data Mining (SDM)*, 587–592.

Shah, N. B.; Bradley, J. K.; Parekh, A.; Wainwright, M.; and Ramchandran, K. 2013. A case for ordinal peer-evaluation in MOOCs. In *Neural Information Processing Systems (NIPS): Workshop on Data Driven Education*.

Warren, H. 1968. Lower bounds for approximation by nonlinear manifolds. *Transaction of the American Mathematical Society* 133:167–178.