Optimizing positional scoring rules for rank aggregation*

Ioannis Caragiannis Xenophon Chatzigeorgiou George A. Krimpas

Department of Computer Engineering and Informatics, University of Patras, Greece

Alexandros A. Voudouris

Department of Computer Science, University of Oxford, UK

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 viewpoint and present positive and negative complexity results and, furthermore, complement our theoretical findings with experiments on real-world and synthetic data.

1 Introduction

Social choice theory [8] 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; the papers [16, 18, 39] are some notable exceptions.

Our adoption of incomplete rankings is motivated by *crowdsourcing* [25] 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

*This work was partially supported by Caratheodory research grant E.114 from the University of Patras, by a PhD scholarship from the Onassis Foundation, and by the European Research Council (ERC) under grant number 639945 (ACCORD).

(i.e., for a full ranking of them) could be problematic since, 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 fixed sets of alternatives. The requester could give each worker a different set of few alternatives to rank. Then, processing smaller inputs and merging them to come up with a global ranking of all alternatives would be easier for the requester as well.

This approach has been recently exploited in the context of ordinal peer grading in massive open online courses (MOOCs); see the papers [3, 10, 11, 38, 40, 41] for approaches of this flavor. 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 of other students to rank, and the final grading (a ranking of all students) is obtained by aggregating the inputs provided by the students. Ordinal peer grading has also been used in the evaluation of proposals for research funding, e.g., by the Sensors and Sensing Systems (SSS) program of NSF in 2013 [22], using a Borda-like method proposed earlier by Merrifield and Saari [34] (see also [24]).

An example of the rating application that we envision is as follows. Users of a hotel booking system are asked to rank hotels in a specific city, in which they have recently stayed. The goal of the system is to compute a full ranking of the hotels (or, possibly, different rankings depending on different relevant criteria, such as price, cleanliness, location, etc.) that can help new users. 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. Similar examples of rating applications include systems related to ranking restaurants, universities, and so on.

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; this is consistent to the ordinal peer grading approach as it has been applied in MOOCs [11, 38] or in the NSF pilot [22]. 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.

The input of our problem consists of 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). Given this input, 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. We refer to this seemingly fundamental optimization problem as OptPSR, standing for "Optimizing Positional Scoring Rules".

1.1 Our contribution

Our technical contribution consists of theoretical and experimental results for OptPSR. We begin by presenting an exact algorithm, which we call `Regions`, that solves OptPSR in time that depends exponentially only on the parameter d. Hence, `Regions` runs in polynomial time when d is constant. For instances with high values of d, we have two approximation algorithms. The first one, which we call `BestApproval`, searches among the class of t-approval voting rules (that use scoring vectors with t 1s followed by $d - t$ 0s, with $t \in [d]$) and returns the one that satisfies constraints of highest total weight. The solutions returned by `BestApproval` are always at least $1/d$–approximate. This means that the total weight of the satisfied constraints is at least $1/d$ times the maximum total weight of constraints that can be simultaneously satisfied by any scoring rule. We show that our analysis is tight by constructing simple instances, in which any approval voting rule is (at most) $1/d$-approximate. We also present a second, more sophisticated, approximation algorithm, called `ApxPSR`, which achieves even better approximation ratios at the expense of higher (but still polynomial) running time. On the negative side, we show that OptPSR is not only computationally hard (in particular, NP-hard) to compute exactly but also NP-hard to approximate. We present an explicit inapproximability bound of 23/24; our proof is based on an approximation-preserving reduction from the optimization 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 [21].

Further, we describe experiments from the execution of scoring rules/algorithms on many OptPSR instances. We use two real-world profiles, which we have carefully collected, as well as numerous synthetic profiles that are produced by simulating agents whose ranking behavior follows the Bradley-Terry [7] and Plackett-Luce [29, 36] noise models. In contrast to our theoretical work, which is based on worst-case assumptions, our experimental results show that well-known scoring rules as well as our algorithms `ApxPSR` and `BestApproval` perform remarkably well and recover almost 100% of the desired constraints in all scenarios we examined; this justifies our choice to study the optimization problem OptPSR in the first place.

1.2 Related work

Social choice theory has traditionally assumed that voters provide full rankings (strict linear orders) over all alternatives. There are some deviations from this tradition that have been attempted recently. Among other issues, Boutilier and Rosenschein [6] discuss settings with incomplete rankings as votes. In general, these models belong to one of the following categories. Several papers (see, e.g., [4, 17, 20]) consider voters who rank their few top preferred alternatives. Others, like the current paper, consider voters who rank arbitrary subsets of alternatives. These include the papers [16, 18, 39], in which voters rank alternatives they know. For example, in [18], the alternatives are web pages and the voters are different search engines, which do not necessarily have full coverage of the web. In the papers on ordinal peer grading mentioned above [3, 10, 11, 24, 34, 38, 40, 41], the "voters" are asked to rank particular subsets of alternatives. In this sense, even though they are in

principle capable of evaluating other alternatives, they are never asked to do so and, hence, do not include them in their rankings. In the most general setting, each vote can be a *partial order* of all alternatives. This includes the cases mentioned above as well as the case in which some relations between pairs of alternatives are not given. Konczak and Lang [23] were the first to consider this setting in computational social choice, followed by Pini et al. [35], Xia and Conitzer [45], and others.

Important problems in the setting with partial orders of alternatives as votes are related to how the votes can be extended to full rankings. The papers [23, 35, 45] mentioned above consider the question of whether there exists an extension of a profile of partial orders so that a given alternative becomes the winner. The question of whether a given alternative is the winner in *any* profile extension has also been studied. Both questions give rise to elegant computational problems, which are informally known as *possible* and *necessary winners*, respectively. Lu and Boutilier [27] view the selection among different possible winners as a robust optimization problem and use the notion of minimax regret to solve it. Their techniques extend to multi-winner determination [28]. However, all these papers focus on the winning alternative(s), while our interest here is on the whole ranking returned by a voting rule that is applied on profiles with incomplete votes.

The existence of an underlying true ranking is a central assumption in our work. This is closely related to a trend in social choice, which assumes an objectively correct ranking of the alternatives (a *ground truth*) and views votes as noisy estimates of this ranking. The most common approach in such settings aims to view voting rules as *maximum likelihood estimators* [14, 46] and start with the assumption that each voter implicitly transforms the ground truth into her vote following a particular probability distribution or noise model. A voting rule is an MLE for a noise model if, when applied on a profile of votes, it returns as an outcome the ranking or the winning alternative that is the most likely to produce this profile, assuming that voters follow the noise model. The most prominent such result is due to Young [46], who proved that the Kemeny voting rule is the MLE for a noise model that dates back to Marquis de Condorcet [15] and is today better known as the Mallows' model [30]. A discussion on recent results on the MLE approach can be found in the chapter by Elkind and Slinko [19]. Among them, Xia and Conitzer [44] and Lu and Boutilier [26] use the MLE approach to voting rules applied on profiles with incomplete votes. A related line of research aims to establish sample complexity results. How many votes (from a given noise model) are necessary in order to recover the ground truth with high probability? The papers [9, 12, 13] follow this direction.

Another approach, which is even closer to the current paper, has an optimization flavor. The papers [10, 11] of our group on ordinal peer grading as well as the paper by de Weerdt et al. [16] aim to identify the voting rule whose outcome ranking has as small expected distance from the ground truth ranking as possible. In general, these voting rules are not maximum likelihood estimators. In contrast to these papers as well as to those following the MLE approach, here we assume that we have access to parts of the underlying ground truth. Furthermore, in our theoretical investigations, we do not exploit the fact that votes may be noisy estimates of some ground truth but, instead, treat them as arbitrary; this gives rise to several optimization challenges. On the other hand, our experimental scenarios use ground truth rankings and voters (agents) that follow two well-known noise models.

Finally, our optimization problem OptPSR aims to compute the positional scoring rule that best fits the input. This is conceptually related to learning-theoretic studies that seek a scoring rule that is consistent to given examples; e.g., see the papers by Boutilier et al. [5] and Procaccia et al. [37] which, among other results, study the sample complexity of scoring rules by bounding their

generalized dimension. Like our theoretical investigations, their settings are more general and do not depend on particular noise models.

1.3 Roadmap

The rest of the paper is structured as follows. We begin with the formal description of the OptPSR problem and necessary preliminary material in Section 2, including definitions, an example, as well as the description of a naive exact algorithm for OptPSR. In Section 3, we present and analyze our exact algorithm `Regions`. Our approximation algorithms `BestApproval` and `ApxPSR` are presented and analyzed in Section 4. In Section 5, we give the proof of our inapproximability result for OptPSR. Our experiments follow in Section 6. We conclude with a discussion on open problems and possible extensions in Section 7. Additional material related to our experiments is given in Appendix.

2 Problem statement

We consider settings with a set N of *agents* and a set A of *alternatives*. Agent i expresses her preference over a subset $A_i \subseteq A$ of alternatives; her preference (or vote) is a strict linear order (henceforth, simply, a *ranking*) of the alternatives in A_i. A preference profile (or simply, a *profile*) Π 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 preferences, i.e., $|A_i| = d$ for each agent i.

A social welfare function takes as input a profile Π 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*). In the following, with some abuse in notation, we use $\mathbf{s}$ to refer to both the scoring vector and the corresponding scoring rule that uses it. Formally, for an alternative x, let $\nu_j(x, \Pi)$ denote the number of agents that rank x at position j in profile Π. 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 remark that this score definition does not take into account the popularity of alternative x (i.e., the number of times x appears in the rankings of the profile). Another definition would normalize the score by dividing with the number of appearances of x in the profile. We have chosen the current definition for proof-of-concept purposes only.

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) 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 Π 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 we will simply refer to it as the profile. Each constraint $(x, y) \in C$ has a corresponding non-negative weight $w(x, y)$, which indicates the importance of the constraint.

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)

$$\mathsf{g}(\mathbf{s}, \boldsymbol{\delta}(\Pi), C) = \sum_{(x,y)\in C} w(x, y) \cdot \mathbb{1}\left\{\boldsymbol{\delta}(x, y, \Pi) \cdot \mathbf{s} > 0\right\},$$

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

Example 1. Consider ten agents, a set of seven alternatives $A = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$, the profile of rankings of size $d = 4$ that appears in Table 1, and the constraints that appear in Table 2 together with the corresponding weights and the representation $\boldsymbol{\delta}(\Pi)$ of the profile.

First, observe that the first three constraints cannot be satisfied simultaneously; this can be easily seen since by summing the corresponding inequalities we obtain $-s_2 + s_3 - 8s_4 > 0$ which contradicts $s_2 \geq s_3$ and $s_4 \geq 0$. So, in the best case, the optimal scoring rule can satisfy all constraints besides the one among the first three that has the minimum weight. One such scoring rule uses the scoring vector $(4, 4, 1, 0)$ and satisfies all constraints except the second one for a total gain of 10. In contrast, well-known scoring rules such as the Borda count that uses the scoring vector $(3, 2, 1, 0)$ and the t-approval rules, with $t \in [4]$, that use scoring vectors with t ones followed by $4 - t$ zeros, satisfy constraints of total gain equal to 7, 8 (for $t = 1$), 8 (for $t = 2$), 9 (for $t = 3$), and 4 (for $t = 4$), respectively; see also Table 3. □

# of agents	ranking
1	$x_1 \succ x_2 \succ x_6 \succ x_4$
2	$x_3 \succ x_1 \succ x_4 \succ x_2$
2	$x_5 \succ x_6 \succ x_4 \succ x_2$
1	$x_7 \succ x_3 \succ x_4 \succ x_2$
1	$x_5 \succ x_4 \succ x_3 \succ x_6$
3	$x_7 \succ x_5 \succ x_4 \succ x_2$

Table 1: The profile Π of agent rankings in Example 1. The notation $\succ$ is used here to represent the preference of the agents. For instance, according to the second row of the table, there are two agents that rank alternative x_3 first, alternative x_1 second, x_4 third, and x_2 last.

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

constraint	weight	$\delta_1(\cdot,\Pi)$	$\delta_2(\cdot,\Pi)$	$\delta_3(\cdot,\Pi)$	$\delta_4(\cdot,\Pi)$	inequality
(x_1,x_2)	4	1	1	0	-8	$s_1+s_2-8s_4>0$
(x_4,x_5)	2	-3	-2	8	1	$-3s_1-2s_2+8s_3+s_4>0$
(x_3,x_4)	3	2	0	-7	-1	$2s_1-7s_3-s_4>0$
(x_4,x_6)	2	0	-1	7	0	$-s_2+7s_3>0$
(x_3,x_2)	1	2	0	1	-8	$2s_1+s_3-8s_4>0$

Table 2: The constraints, the corresponding weights, the alternative representation of the profile Π using the quantities $\boldsymbol{\delta}(x,y,\Pi)$, and the induced inequalities used in Example 1. For instance, the first constraint of weight 4 requires that alternative x_1 appears above x_2 in the final ranking. According to the profile Π in Table 1, since there are two agents that place alternative x_1 in the second position, while there is only one agent that places alternative x_2 in the second position, we have that $\delta_2(x_1,x_2,\Pi)=1$; one can easily verify the remaining values of the table. Given these quantities, the inequalities follow since $\delta_j(x_1,x_2,\Pi)$ for $j\in[4]$ is the coefficient of the variable s_j corresponding to the points assigned to position j.

rule	scoring vector	gain
opt	$(4,4,1,0)$	10
Borda count	$(3,2,1,0)$	7
1-approval	$(1,0,0,0)$	8
2-approval	$(1,1,0,0)$	8
3-approval	$(1,1,1,0)$	9
4-approval	$(1,1,1,1)$	4

Table 3: The positional scoring rules considered in Example 1 and their corresponding total gains.

partition R_0 into disjoint subregions; of course, some of them may be infeasible. Hence, in order to maximize $\mathsf{g}(\mathbf{s},\boldsymbol{\delta}(\Pi),C)$, 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 practice, the parameter d (i.e., the size of the input rankings) is expected to be a small constant, while the number of alternatives and, consequently, the number of constraints $|C|$ would be much larger. Hence, an algorithm that runs in time exponential in $|C|$ is clearly impractical. 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 which we call `Regions`. Its running time depends exponentially only on the parameter d and, hence, is polynomial when d is a constant. We remark that this time complexity is interesting only in theory. As we will mention later, when describing our experiments in Section 6, even when d is small (say, equal to 6), the algorithm does not scale well with the number of constraints.

`Regions` computes a pool of non-empty subregions of R_0, each of which satisfies a different subset of constraints. Initially, the pool consists only of region R_0, and is updated as new constraints of C are considered. When a new constraint is considered, each region in the pool can be split into two subregions consisting of the points that satisfy the constraint and the points that do not satisfy it, respectively. When all points of a region satisfy the constraint or all points of the region do not satisfy it, then the region is not split and is retained as a whole in the pool.

In particular, the algorithm considers the constraints of C one by one. At each step t, 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 $\texttt{val}(R)$ 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 $\boldsymbol{\delta}(x, y, \Pi) \cdot \mathbf{s} > 0$ (which 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 $\boldsymbol{\delta}(x, y, \Pi) \cdot \mathbf{s} \leq 0$ (which 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 $\texttt{val}(R^{xy}) := \texttt{val}(R) + w(x, y)$ and $\texttt{val}(R^{\neg xy}) := \texttt{val}(R)$, and removes region R from the pool. If only R^{xy} is feasible (and $R^{\neg xy}$ is infeasible), $\texttt{val}(R)$ is increased by $w(x, y)$. If only $R^{\neg xy}$ is feasible (and R^{xy} is infeasible), 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. Again, 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 $\texttt{val}(R^*)$ and returns any scoring vector in R^*. A description of the algorithm in pseudocode is given as Algorithm 1.

Example 2. We will now examine a simple example of how `Regions` works on a profile Π with alternatives x_1, x_2, x_3, y_1, y_2, and y_3, and $d = 2$ (see Figure 1). 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 $\boldsymbol{\delta}(x_1, y_1, \Pi) = (-7, 2)$, $\boldsymbol{\delta}(x_2, y_2, \Pi) = (4, -2)$, and $\boldsymbol{\delta}(x_3, y_3, \Pi) = (-2, 3)$. Therefore, the constraints define the inequalities $-7s_1 + 2s_2 > 0$, $4s_1 - 2s_2 > 0$, and $-2s_1 + 3s_2 > 0$.

Now, the algorithm proceeds as follows. Initially (see Figure 1a), the algorithm has region R_0, defined by the lines $s_2 = 0$ and $s_1 - s_2 = 0$, in the pool with gain equal to 0. At the next step (see Figure 1b), the algorithm considers constraint (x_1, y_1) and replaces R_0 with regions $R_1 = R_0^{x_1 y_1}$ and $R_2 = R_0^{\neg x_1 y_1}$; R_1 is the subregion of R_0 with $-7s_1 + 2s_2 > 0$ that satisfies the first constraint and has gain 3, while R_2 is the subregion of R_0 with $-7s_1 + 2s_2 < 0$ that does not satisfy the first constraint and has gain 0 (the line $-7s_1 + 2s_2 = 0$ separates the two subregions). Next (see Figure 1c), the constraint (x_2, y_2) leaves both regions R_1 and R_2 in the pool, and increases both of their gains by 1. Finally (see Figure 1d), the third constraint (x_3, y_3) replaces region R_1 by regions $R_3 = R_1^{x_3 y_3}$ and $R_4 = R_1^{\neg x_3 y_3}$; R_3 is the subregion of R_1 with $-2s_1 + 3s_2 > 0$ that satisfies the constraint and has gain equal to 6 (the gain of R_1 increased by the weight of the constraint), while

Algorithm 1: `Regions`

```
Input: parameter d, profile δ(Π), set C of constraints
Output: a scoring vector s = (s_1, ..., s_d)
R_0 := {s_1 ≥ s_2, ..., s_{d-1} ≥ s_d, s_d ≥ 0}
val(R_0) := 0
P := {R_0}
for (x, y) ∈ C do
    active := P
    for R ∈ active do
        active := active \ {R}
        R^{xy} := {R, δ(x, y, Π) · s > 0}
        R^{¬xy} := {R, δ(x, y, Π) · s ≤ 0}
        if R^{xy} is feasible and R^{¬xy} is feasible then
            val(R^{xy}) := val(R) + w(x, y)
            val(R^{¬xy}) := val(R)
            P := {P \ {R} ∪ {R^{xy}, R^{¬xy}}}
        else if R^{xy} is feasible and R^{¬xy} is not feasible then
            val(R) := val(R) + w(x, y)
        end
    end
end
R* := arg max_{R∈P}{val(R)}
return s ∈ R*
```

R_4 is the subregion of R_1 with $-2s_1 + 3s_2 < 0$ that does not satisfy the constraint and has gain 4 (equal to that of R_1). Observe that in the last step, region R_2 also satisfies the third constraint and, hence, its gain is also increased by 2. The region with the maximum gain is R_3 and the algorithm will output some scoring vector from this region. □

Next, we prove that our improved OptPSR algorithm is correct and that its running time depends exponentially only on the parameter d.

Theorem 1. *Given an instance of* OptPSR *with parameter d, a set of constraints C, and a profile Π, algorithm* `Regions` *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 1e 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

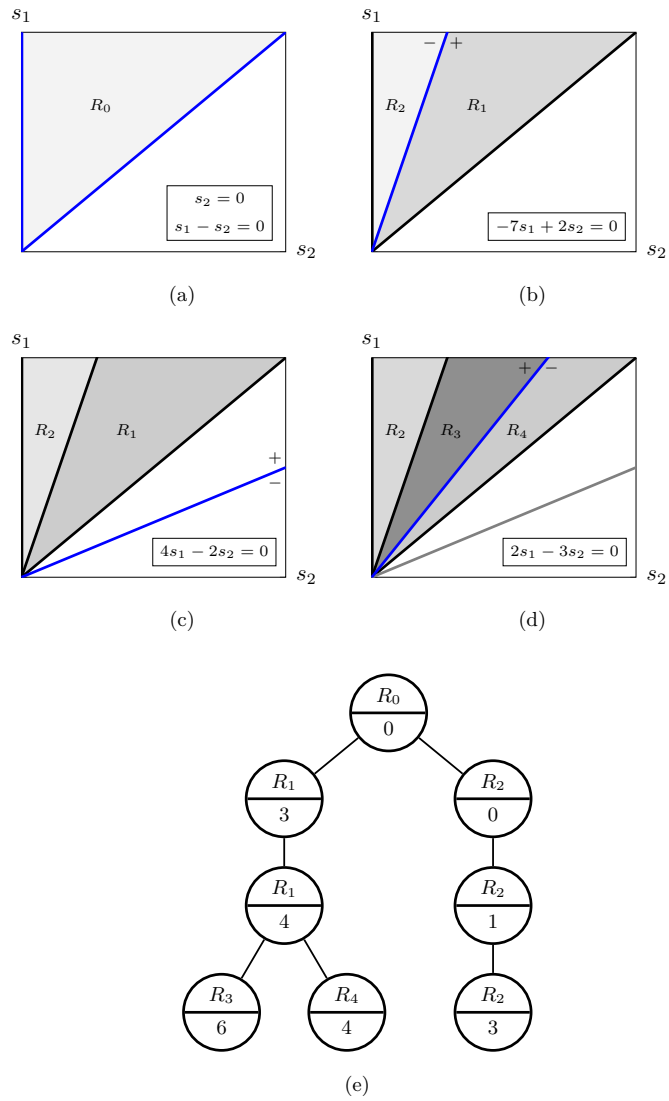


Figure 1: An example with the execution of `Regions` on a profile Π with $d = 2$. Subfigures (a)–(d) depict the step-by-step consideration of the constraints and how the active regions are updated. At each subfigure, the blue line corresponds to a new constraint that is considered. In Subfigures (b)–(d), the marks $+$ and $-$ denote which of the two subregions defined by the blue line contain the vectors that satisfy the constraint or not. The gain of the subregions marked with $+$ are increased by the weight of the constraint (a darker shade of gray for a subregion indicates a higher gain). Subfigure (e) depicts the evolution of the content of the pool of regions together with the corresponding gains. Specifically, the nodes in each level of the tree represent the content of the pool and the corresponding gains during each step of the algorithm.

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 $\boldsymbol{\delta}(x, y, \Pi) \cdot \mathbf{s}$ 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 [1] (see also Warren [43]) 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 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. *Algorithm* `Regions` *solves instances of* OptPSR *with constant d in polynomial time.*

4 Approximating OptPSR

As the running time of the exact algorithm `Regions` 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. Given an instance of OptPSR with parameter d, profile Π, and set C of constraints, let $\mathbf{s}^*$ be the scoring vector that satisfies constraints of C with maximum total weight. A scoring vector $\mathbf{s}$ is a ρ-approximate solution, for some $\rho \in [0, 1]$, for the particular instance if $\mathsf{g}(\mathbf{s}, \boldsymbol{\delta}(\Pi), C) \geq \rho \cdot \mathsf{g}(\mathbf{s}^*, \boldsymbol{\delta}(\Pi), C)$, i.e., the total weight of constraints satisfied by $\mathbf{s}$ is at least ρ times the total weight of constraints satisfied by $\mathbf{s}^*$. An algorithm is called a ρ-approximation algorithm if it computes a ρ-approximate solution for every instance of OptPSR. We refer to ρ as the *approximation ratio* of the algorithm. Ideally, we would like to design approximation algorithms that are as efficient as possible, i.e., algorithms that run in polynomial-time and have as high approximation ratio as possible.

Let us warm up by observing that a single positional scoring rule (e.g., Borda, plurality, k-approval) cannot serve as an efficient approximation algorithm as it has an approximation ratio of 0. This is stated in the next lemma.

Lemma 3. *Let d be a positive integer. For every scoring vector $\mathbf{s} \in \mathbb{R}^d_{\geq 0}$ with $s_1 \geq ... \geq s_d$, there exists an instance of* OptPSR *with parameter d, profile Π and set C of constraints such that $\mathbf{s}$ is 0-approximate.*

Proof. Clearly, the scoring vector $\mathbf{s} = (0, ..., 0)$ does not satisfy any constraint. So, in the following, we assume that $s_1 > 0$. For any positive integer $K > 0$, we will construct a set C of constraints consisting of disjoint pairs of alternatives (x_t, y_t) for $t = 1, ..., K$. We will distinguish between two cases depending on the structure of $\mathbf{s}$. For each of them we will construct an OptPSR instance (formed by an appropriately defined profile and the set C of constraints), in which $\mathbf{s}$ satisfies no constraint, while another scoring vector $\mathbf{s}^*$ satisfies all of them.

If $s_1 = ... = s_d > 0$, then for every constraint (x_t, y_t) we set $\delta_1(x_t, y_t, \Pi) = 1$, $\delta_2(x_t, y_t, \Pi) = -2$ and $\delta_j(x_t, y_t, \Pi) = 0$ for $j = 3, ..., d$. Profile Π can be realized as follows. Alternative x_t appears in position 1 once and alternative y_t appears in position 2 twice; all other positions are filled with additional alternatives that do not appear in the constraints of C. Observe that $\boldsymbol{\delta}(x_t, y_t, \Pi) \cdot \mathbf{s} =$

$s_1 - 2s_2 < 0$ for every $t = 1, ..., K$, i.e., $\mathbf{s}$ does not satisfy any constraint. In contrast, the plurality vector $\mathbf{s}^* = (1, 0, ..., 0)$ satisfies all constraints of C.

Otherwise, if $s_i > s_{i+1}$ for some $i \in [d-1]$, let D be an integer satisfying $D > \frac{s_{i+1}}{s_i - s_{i+1}}$. For every constraint (x_t, y_t) for $t = 1, ..., K$, we set $\delta_i(x_t, y_t, \Pi) = -D$, $\delta_{i+1}(x_t, y_t, \Pi) = D + 1$, and $\delta_j(x_t, y_t, \Pi) = 0$ for $j \in [d] \setminus \{i, i+1\}$. Profile Π can be realized as follows. Alternative x_t appears D times in position i and alternative y_t appears $D + 1$ times in position $i + 1$; again, all other positions are filled with additional alternatives that do not appear in the constraints of C. Observe that the definition of D implies that $\boldsymbol{\delta}(x_t, y_t, \Pi) \cdot \mathbf{s} = -s_i D + s_{i+1}(D + 1) < 0$, i.e., $\mathbf{s}$ does not satisfy any constraint. In contrast, the $(i+1)$-approval vector $\mathbf{s}^*$ (consisting of 1s in the first $i+1$ positions and 0s in the remaining ones) satisfies all constraints of C.

In both cases, the scoring vector $\mathbf{s}$ is a 0-approximate solution. □

We conclude that efficient approximation algorithms should consider many candidate scoring vectors and pick the one that better serves a given OptPSR instance. This is a recipe that is followed by the algorithms `BestApproval` and `ApxPSR` that are presented in Sections 4.1 and 4.2, respectively.

4.1 Approximating OptPSR using approval scoring vectors

We will now show that an extremely simple algorithm that examines a set of simple scoring vectors and returns the best of them achieves a $1/d$-approximation solution. Formally, 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. Our algorithm, which we call `BestApproval`, considers all t-approval rules and returns the one that satisfies constraints of maximum total weight. A description of `BestApproval` in pseudocode is given as Algorithm 2.

Algorithm 2: `BestApproval`

Input: parameter d, profile $\boldsymbol{\delta}(\Pi)$, set C of constraints
Output: a t^*-approval rule
for $t \in [d]$ **do**
  $\mathbf{t} := (\underbrace{1, ..., 1}_{t \text{ times}}, 0, ..., 0)$
  $\mathtt{g}(\mathbf{t}, \boldsymbol{\delta}(\Pi), C) := \sum_{(x,y) \in C} w(x, y) \cdot \mathbb{1}\{\boldsymbol{\delta}(x, y, \Pi) \cdot \mathbf{t} > 0\}$
end
return $t^* \in \arg\max_t \{\mathtt{g}(\mathbf{t}, \boldsymbol{\delta}(\Pi), C)\}$

With the following two theorems, we prove that `BestApproval` is a $1/d$-approximation algorithm for OptPSR (Theorem 4) and, furthermore, we show that this bound is tight by providing a particular instance for which any t-approval rule is an (at most) $1/d$-approximate solution (Theorem 5).

Theorem 4. *Given an instance of* OptPSR *with parameter d, a set of constraints C, and a profile Π, algorithm* `BestApproval` *returns a $1/d$-approximate solution.*

Proof. In order to prove the bound on the approximation ratio, we will show that there exists some $t \in [d]$ so that the corresponding t-approval scoring rule is a $1/d$-approximate solution. Then, the t^*-approval rule returned by `BestApproval` will have an at least as good approximation guarantee.

Let $\mathbf{s}^* \in \mathbb{R}^d_{\geq 0}$ be the optimal scoring rule for the given instance of OptPSR, and let $X \subseteq C$ be the set of constraints that it satisfies, i.e., $\boldsymbol{\delta}(x, y, \Pi) \cdot \mathbf{s}^* > 0$ if and only if $(x, y) \in X$. For $k \in [d]$, let

$$X_k = \left\{ (x, y) \in X : \sum_{j=1}^{k} \delta_j(x, y, \Pi) > 0 \right\}.$$

I.e., X_k contains the constraints of X which are satisfied by the k-approval scoring rule. This definition implies that

$$\sum_{k=1}^{d} \mathsf{g}(\mathbf{k}, \boldsymbol{\delta}(\Pi), C) \geq \sum_{k=1}^{d} \sum_{(x,y) \in X_k} w(x, y). \tag{1}$$

We now claim that each constraint $(x, y) \in X$ belongs to X_k for some $k \in [d]$ and, hence,

$$\sum_{(x,y) \in X} w(x, y) \leq \sum_{k=1}^{d} \sum_{(x,y) \in X_k} w(x, y). \tag{2}$$

Assume otherwise; then, it would mean that $\sum_{j=1}^{k} \delta_j(x, y, \Pi) \leq 0$ for every $k \in [d]$. By multiplying these d inequalities with the non-negative quantities $s^*_k - s^*_{k+1}$ for $k = 1, ..., d-1$ and s^*_d, and summing them, we obtain that

$$\sum_{k=1}^{d-1} (s^*_k - s^*_{k+1}) \sum_{j=1}^{k} \delta_j(x, y, \Pi) + s^*_d \sum_{j=1}^{d} \delta_j(x, y, \Pi) \leq 0.$$

The claim follows by observing that the left-hand side is equal to $\boldsymbol{\delta}(x, y, \Pi) \cdot \mathbf{s}^*$ and, hence, $\boldsymbol{\delta}(x, y, \Pi) \cdot \mathbf{s}^* \leq 0$, contradicting the fact that $(x, y) \in X$.

Using (2) and (1), we obtain

$$\mathsf{g}(\mathbf{s}^*, \boldsymbol{\delta}(\Pi), C) = \sum_{(x,y) \in X} w(x, y) \leq \sum_{k=1}^{d} \sum_{(x,y) \in X_k} w(x, y) \leq \sum_{k=1}^{d} \mathsf{g}(\mathbf{k}, \boldsymbol{\delta}(\Pi), C).$$

This implies that there exists $t \in [d]$ such that $\mathsf{g}(\mathbf{t}, \boldsymbol{\delta}(\Pi), C) \geq \frac{1}{d} \mathsf{g}(\mathbf{s}^*, \boldsymbol{\delta}(\Pi), C)$, as desired. The theorem follows. □

Theorem 5. *There exists an instance of* OptPSR *with parameter d for which all t-approval rules, with $t \in [d]$, are (at most) $1/d$-approximate.*

Proof. We will define an OptPSR instance with 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 Π so that the t-approval scoring rule satisfies only constraint (x_t, y_t) for $t \in [d]$, while there exists a scoring rule that simultaneously satisfies all constraints. The profile is defined as follows:

- Alternative x_1 appears $2d - 1$ times in position 1, and alternative y_1 appears $2d$ times in position 2. This means that $\delta_1(x_1, y_1, \Pi) = 2d - 1$, $\delta_2(x_1, y_1, \Pi) = -2d$ and $\delta_j(x_1, y_1, \Pi) = 0$ for $j \geq 3$.

- For $2 \leq t \leq d-1$, alternative x_t appears $2d$ times in position t, and alternative y_t appears once in position 1 and $2d$ times in position $t+1$. This means that $\delta_1(x_t, y_t, \Pi) = -1$, $\delta_t(x_t, y_t, \Pi) = 2d$, $\delta_{t+1}(x_t, y_t, \Pi) = -2d$, and $\delta_j(x_t, y_t, \Pi) = 0$ for $j \notin \{1, t, t+1\}$.
- Alternative x_d appears $2d$ times in position d, and alternative y_d appears once in position 1. This means that $\delta_1(x_d, y_d, \Pi) = -1$, $\delta_d(x_d, y_d, \Pi) = 2d$, 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 every $t \in [d]$, it holds that $\sum_{j=1}^{t} \delta_j(x_t, y_t, \Pi) = 2d - 1 > 0$ and $\sum_{j=1}^{t} \delta_j(x_\ell, y_\ell, \Pi) = -1$, for any $\ell \neq [d] \setminus \{t\}$. Hence, the t-approval scoring rule satisfies only constraint (x_t, y_t) for a total weight of 1. Now, consider a variation of the Borda count scoring rule that uses the scoring vector $\mathbf{s} = (d, d-1, ..., 1)$. Then, since $\boldsymbol{\delta}(x_\ell, y_\ell, \Pi) \cdot \mathbf{s} = d > 0$ for every $\ell \in [d]$, this scoring rule satisfies all constraints for a total weight of d. Therefore, we conclude that any t-approval has approximation ratio at most $1/d$. □

4.2 An improved approximation algorithm

Here, we will design the more sophisticated approximation algorithm $\texttt{ApxPSR}_k$, which is parameterized by a positive integer k, and exploits ideas that we have already presented above. Similarly to `BestApproval`, $\texttt{ApxPSR}_k$ searches over a set of candidate scoring vectors and identifies the one that better serves the available input data (profile and constraints). Two important differences between $\texttt{ApxPSR}_k$ and `BestApproval` are that (a) the set of candidate scoring rules is much broader now and (b) a few executions of a variation of the exact algorithm `Regions` are used in order to find the best among these candidates.

Let $\ell \in [\lceil d/k \rceil]$. We say that a scoring vector $\mathbf{s}$ follows the ℓ-th k-pattern if $s_1 = ... = s_{k(\ell-1)+1} \geq s_{k(\ell-1)+2} \geq ... \geq s_{\min\{d, k\ell\}} \geq 0$ and, if $\ell < \lceil d/k \rceil$, $s_{k\ell+1} = ... = s_d = 0$. For example, the t-approval scoring rule follows that t-th 1-pattern. Note that the scoring vectors that follow some of the $\lceil d/k \rceil$ k-patterns have a very special structure and (at most) k different values in their score entries.

For a given instance of OptPSR and integers $k > 0$ and $\ell \in [\lceil d/k \rceil]$, we can compute the best scoring vector that follows the ℓ-th k-pattern via a slight modification of `Regions`. We refer to this modification as algorithm `mRegions` and we assume that, together with parameter d, the profile $\boldsymbol{\delta}(\Pi)$, and the set C of constraints, it receives as input the parameters k and ℓ as well. All we need to do is to include the equality constraints which restrict the entries of scoring vectors that follow the ℓ-th k-pattern in the initial region R_0. These equality constraints are included in all (sub)regions that are considered by the algorithm. Hence, the regions that will be contained in the pool $\mathcal{P}$ when `mRegions` terminates will all satisfy the equality restrictions. In this way, the scoring vector that will be computed will follow the ℓ-th k-pattern, as desired.

Our algorithm $\texttt{ApxPSR}_k$ first calls `mRegions` to compute the $\lceil d/k \rceil$ best scoring rules that follow the ℓ-th k-pattern for $\ell = 1, ..., \lceil d/k \rceil$ and returns the best among all these rules, i.e., the one that yields the highest gain among them. Algorithm $\texttt{ApxPSR}_k$ follows as Algorithm 3.

The next theorem summarizes the properties of algorithm $\texttt{ApxPSR}_k$. Observe that the algorithm runs in polynomial time when the parameter k is a constant. The approximation ratio is better than approximately k times that of `BestApproval`.

Theorem 6. *Given an instance of* OptPSR *with parameter d, a set of constraints C, and a profile Π, algorithm $\texttt{ApxPSR}_k$ runs in time $\mathcal{O}(|C|^k \cdot \text{poly}(|C|, d))$ and returns a $\lceil d/k \rceil^{-1}$-approximate solution.*

Algorithm 3: $\texttt{ApxPSR}_k$

Input: parameter d, profile $\boldsymbol{\delta}(\Pi)$, set C of constraints
Output: a scoring vector $\mathbf{s} = (s_1, ..., s_d)$
$S := \emptyset$
for $\ell \in [\lceil d/k \rceil]$ **do**
  $S := S \cup \texttt{mRegions}(d, \boldsymbol{\delta}(\Pi), C, k, \ell)$
end
return $\mathbf{s} \in \arg\max_{\mathbf{s}' \in S} \{\mathtt{g}(\mathbf{s}', \boldsymbol{\delta}(\Pi), C)\}$

Proof. We first show the bound on the running time. Observe that $\texttt{ApxPSR}_k$ selects the best scoring vector among those returned in $\lceil d/k \rceil$ executions of `mRegions`. We will show that each execution of `mRegions` takes time $\mathcal{O}(|C|^k \cdot \text{poly}(|C|, d))$.

By mimicking the proof of Theorem 1, we can view the expansion of the pool by `mRegions` as a non-complete binary tree T with nodes corresponding to regions. Then, the total time required to find all regions by `mRegions` is proportional to the size of T, which is at most its height $|C|$ times the number of leaves. The number of leaves is again the number of different non-empty regions, which is upper-bounded by the number of different sign patterns that the quantities $\boldsymbol{\delta}(x, y, \Pi) \cdot \mathbf{s}$ define for each constraint (x, y) in C. Since these $|C|$ quantities are linear functions over k (as opposed to d in the proof of Theorem 1) coordinates of vector $\mathbf{s}$, the results of Alon [1] and Warren [43] yield that the total number of different sign patterns is at most $\left(\frac{8e|C|}{k}\right)^k$. For each of the nodes of T, feasibility can again be checked by solving two linear programs with d variables and at most $|C|$ constraints in time $\text{poly}(|C|, d)$. The bound on the running time follows.

In order to prove the bound on the approximation ratio, we will show that there exists a scoring vector $\mathbf{s}^{(k)}$ that follows some k-pattern which is a $\left\lceil \frac{d}{k} \right\rceil^{-1}$-approximate solution to the OptPSR instance. Then, the scoring vector $\mathbf{s}$ returned by algorithm $\texttt{ApxPSR}_k$ will have the same approximation guarantee, since $\texttt{ApxPSR}_k$ returns the best scoring vector following some k-pattern, i.e., $\mathtt{g}(\mathbf{s}, \boldsymbol{\delta}(\Pi), C) \geq \mathtt{g}(\mathbf{s}^{(k)}, \boldsymbol{\delta}(\Pi), C)$.

Consider an instance of OptPSR consisting of a profile Π and a set of constraints C with weighting $w : C \to \mathbb{R}_{\geq 0}$. Let $\mathbf{s}^* \in \mathbb{R}^d_{\geq 0}$ be the optimal scoring vector for this instance. We will show that there exists a scoring vector $\mathbf{s}^{(k)}$ that follows some k-pattern such that $\mathtt{g}(\mathbf{s}^{(k)}, \boldsymbol{\delta}(\Pi), C) \geq \left\lceil \frac{d}{k} \right\rceil^{-1} \mathtt{g}(\mathbf{s}^*, \boldsymbol{\delta}(\Pi), C)$.

Let $X \subseteq C$ be the set of constraints that are satisfied by the scoring vector $\mathbf{s}^*$. We now define an alternative view of $\mathbf{s}^*$ by setting $\alpha_d = s_d$ and $\alpha_i = s^*_i - s^*_{i+1}$ for $i = 1, ..., d-1$. I.e., instead of keeping the entries of the scoring vector $\mathbf{s}^*$, we use the vector $\boldsymbol{\alpha} = (\alpha_1, ..., \alpha_d)$ to represent the entry s^*_d and the increase of s^*_i compared to s^*_{i+1} for $i = 1, ..., d-1$. Hence, $s^*_j = \sum_{i=j}^d \alpha_i$ for $j = 1, ..., d$. Using the definition of $\boldsymbol{\delta}(x, y, \Pi)$, for every $(x, y) \in X$, we have

$$\boldsymbol{\delta}(x, y, \Pi) \cdot \mathbf{s}^* = \sum_{j=1}^d \delta_j(x, y, \Pi) \cdot s^*_j = \sum_{j=1}^d \delta_j(x, y, \Pi) \sum_{i=j}^d \alpha_i = \sum_{i=1}^d \alpha_i \sum_{j=1}^i \delta_j(x, y, \Pi).$$

Now, define

$$\xi_\ell(x, y, \Pi) = \sum_{i=(\ell-1)k+1}^{\min\{d, \ell k\}} \alpha_i \sum_{j=1}^i \delta_j(x, y, \Pi)$$

for $\ell \in [\lceil d/k \rceil]$, and observe that the last two equalities imply that

$$\boldsymbol{\delta}(x, y, \Pi) \cdot \mathbf{s}^* = \sum_{\ell=1}^{\lceil d/k \rceil} \xi_\ell(x, y, \Pi). \tag{3}$$

For $\ell \in [\lceil d/k \rceil]$, define the set

$$X_\ell = \{(x, y) \in X : \xi_\ell(x, y, \Pi) > 0\}$$

and observe that, for every constraint $(x, y) \in X$, (3) and the fact that $\boldsymbol{\delta}(x, y, \Pi) \cdot \mathbf{s}^* > 0$ imply that $\xi_\ell(x, y, \Pi) > 0$ for some $\ell \in [\lceil d/k \rceil]$. This yields that

$$\mathrm{g}(\mathbf{s}^*, \boldsymbol{\delta}(\Pi), X) = \sum_{(x,y)\in X} w(x, y) \leq \sum_{\ell=1}^{\lceil d/k \rceil} \sum_{(x,y)\in X_\ell} w(x, y).$$

We conclude that there exists $\ell^* \in [\lceil d/k \rceil]$ such that

$$\sum_{(x,y)\in X_{\ell^*}} w(x, y) \geq \lceil d/k \rceil^{-1} \mathrm{g}(\mathbf{s}^*, \boldsymbol{\delta}(\Pi), X) = \lceil d/k \rceil^{-1} \mathrm{g}(\mathbf{s}^*, \boldsymbol{\delta}(\Pi), C). \tag{4}$$

In order to complete the proof, we will define a particular scoring vector $\mathbf{s}^{(k)}$ that follows a k-pattern and satisfies all constraints in X_{ℓ^*}. We do so by first defining the difference vector $\boldsymbol{\alpha}^{(k)} = (\alpha_1^{(k)}, ..., \alpha_d^{(k)})$ corresponding to $\mathbf{s}^{(k)}$. This is done as follows:

- If $\ell^* > 1$, we set $\alpha_i^{(k)} = 0$ for $i = 1, ..., k(\ell^* - 1)$.
- If $\ell^* < \lceil d/k \rceil$, we set $\alpha_i^{(k)} = 0$ for $i = k\ell^* + 1, ..., d$.
- We set $\alpha_i^{(k)} = \alpha_i$ for $i = k(\ell^* - 1) + 1, ..., \min\{d, k\ell^*\}$.

Now, define the scoring vector $\mathbf{s}^{(k)}$ as $s_j^{(k)} = \sum_{i=j}^d \alpha_i^{(k)}$ for $j = 1, ..., d$. Then, for every $(x, y) \in X$,

$$\begin{aligned}
\boldsymbol{\delta}(x, y, \Pi) \cdot \mathbf{s}^{(k)} &= \sum_{j=1}^d \delta_j(x, y, \Pi) \cdot s_j^{(k)} = \sum_{j=1}^d \delta_j(x, y, \Pi) \sum_{i=j}^d \alpha_i^{(k)} \\
&= \sum_{i=1}^d \alpha_i^{(k)} \sum_{j=1}^i \delta_j(x, y, \Pi) = \sum_{i=k(\ell^*-1)+1}^{\min\{d,k\ell^*\}} \alpha_i^{(k)} \sum_{j=1}^i \delta_j(x, y, \Pi) \\
&= \xi_{\ell^*}(x, y, \Pi).
\end{aligned}$$

This equality, together with the definition of set X_ℓ yields that $\boldsymbol{\delta}(x, y, \Pi) \cdot \mathbf{s}^{(k)} > 0$ for every $(x, y) \in X_{\ell^*}$, i.e., the scoring vector $\mathbf{s}^{(k)}$ satisfies all constraints in X_{ℓ^*}. We obtain that

$$\mathrm{g}(\mathbf{s}^{(k)}, \boldsymbol{\delta}(\Pi), C) \geq \mathrm{g}(\mathbf{s}^{(k)}, \boldsymbol{\delta}(\Pi), X_{\ell^*}) = \sum_{(x,y)\in X_{\ell^*}} w(x, y) \geq \lceil d/k \rceil^{-1} \mathrm{g}(\mathbf{s}^*, \boldsymbol{\delta}(\Pi), C).$$

The last inequality follows by (4). This completes the proof of the approximation bound. □

5 Hardness of approximation

We devote this section to proving that OptPSR is not simply computationally hard, but also hard to approximate well. The proof of our next statement relies on an approximation-preserving reduction from a well-known inapproximable optimization problem.

Theorem 7. *For every constant $\eta \in (0, 1/24]$,* OptPSR *is NP-hard to approximate within* $23/24+\eta$.

Proof. 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. The objective is to find an assignment to the variables so that the number of satisfied equations is maximized. Below, we use the term α-equation to refer to an equation of the form $x_i \oplus x_j \oplus x_k = \alpha$ (for $\alpha \in \{0, 1\}$).

Given an instance of MAX-3LIN-2, our reduction constructs in polynomial-time an instance of OptPSR that has a scoring vector 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 [21] states that, for any constant $\eta' \in (0, 1/2)$, 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. As a consequence of our reduction, we obtain that it is hard to distinguish between instances of OptPSR that have a scoring vector 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 vector is at most $(23/2 + \eta')m$. An inapproximability bound of $23/24+\eta$ (for every constant $\eta \in (0, 1/24]$) then follows by standard arguments. The rest of the proof consists of two main parts: the description of the reduction and the proof of correctness.

Description of the reduction 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 $s_{i+1} = s_i - a_i$ (or, consequently, $s_j = d - \sum_{k=1}^{j-1} a_k$) for $i = 1, ..., d-1$ so that $\sum_{i=1}^{d-1} a_i \leq d$. Hence, a constraint (y, z) requiring that the score of y is higher than the score of z 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.

Our reduction will be described in two steps. Given the MAX-3LIN-2 instance (i.e., the binary variables and the equations), we will first define linear inequalities among the variables a_i. Later, we will define the OptPSR instance by explicitly constructing the profile and specifying the constraints as pairs of alternatives and corresponding weights that are consistent to these linear inequalities.

For the first step of the description of our reduction, we present the linear inequalities that will later evolve to constraints of the OptPSR instance. We set ϵ to be a small constant such that $0 < \epsilon \leq 1/d$ and $1/\epsilon$ is an integer. We define the following inequalities:

- For every variable x_i, we have the four inequalities $a_i > 0$, $a_i < \epsilon$, $a_i > 1$ and $a_i < 1 + \epsilon$. An important property of the four inequalities corresponding to variable x_i is that three of them are simultaneously satisfied when $a_i \in (0, \epsilon) \cup (1, 1 + \epsilon)$, and only two of them are satisfied for any other value of a_i. Intuitively, by enforcing these inequalities to be satisfied as constraints with high weight will simulate binary assignments in the MAX-3LIN-2 instance.

- For every equation, there are four inequalities.
  - 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 $a_i + a_j + a_k < 2 + \epsilon$. Now, three of the inequalities corresponding to a 0-equation $x_i \oplus x_j \oplus x_k = 0$ are simultaneously satisfied when $a_i + a_j + a_k \in (0, \epsilon) \cup (2, 2 + \epsilon)$; otherwise, exactly two inequalities are satisfied.
  - 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 $a_i + a_j + a_k < 3 + \epsilon$. Again, for every 1-equation $x_i \oplus x_j \oplus x_k = 1$, we have three inequalities that are simultaneously satisfied when $a_i + a_j + a_k \in (1, 1+\epsilon) \cup (3, 3+\epsilon)$; otherwise, exactly two inequalities are satisfied.

  These inequalities will correspond to constraints of equal (and low) weight of the OptPSR instance. Overall, our construction will guarantee that the total weight of all satisfied constraints will be linear to the number of satisfied equations in the MAX-3LIN-2 instance. This will be crucial in proving the correctness (and the approximation-preserving nature) of the reduction.

We are now ready to proceed with the second step of the description of the reduction. In particular, we show how the above inequalities are implemented by explicitly constructing a profile and pairs of alternatives as constraints. Let $d = n + 1$. Also, for $i \in [d - 1]$, let m_i be the number of equations in which variable x_i participates. For every variable x_i, with $i \in [n]$, we have four constraints (y_i^t, z_i^t), where $t \in \{1, 2, 3, 4\}$, of weight m_i each. In the profile, alternatives y_i^t and z_i^t appear in specific positions as follows:

- Alternative y_i^1 appears once in position i, and alternative z_i^1 appears once in position $i + 1$. Then, the constraint (y_i^1, z_i^1) corresponds to the inequality $s_i - s_{i+1} > 0$ or, equivalently, $a_i > 0$.

- Alternative y_i^2 appears once in position 1 and d/ϵ times in position $i + 1$, and alternative z_i^2 appears d/ϵ times in position i. The constraint (y_i^2, z_i^2) corresponds to the inequality $s_1 - \frac{d}{\epsilon} s_i + \frac{d}{\epsilon} s_{i+1} > 0$ or, equivalently, $a_i < \epsilon$ (since $s_1 = d$ and $a_i = s_i - s_{i+1}$).

- Alternative y_i^3 appears d times in position i, and alternative z_i^3 appears once in position 1 and d times in position $i+1$. The constraint (y_i^3, z_i^3) corresponds to the inequality $-s_1 + ds_i - ds_{i+1} > 0$ or, equivalently, $a_i > 1$.

- Alternative y_i^4 appears $1/\epsilon + 1$ times in position 1 and d/ϵ times in position $i + 1$, and alternative z_i^4 appears d/ϵ times in position i. The constraint (y_i^4, z_i^4) corresponds to the inequality $\left(\frac{1}{\epsilon} + 1\right) s_1 - \frac{d}{\epsilon} s_i + \frac{d}{\epsilon} s_{i+1} > 0$ or, equivalently, $a_i < 1 + \epsilon$.

For every equation $\ell \in [m]$, we have four constraints (b_ℓ^t, c_ℓ^t), where $t \in \{1, 2, 3, 4\}$, of unit weight each. In the profile, these alternatives appear in specific positions depending on whether equation ℓ is a 0- or a 1-equation. In the case where it is a 0-equation of the form $x_i \oplus x_j \oplus x_k = 0$, we have:

- Alternative b_ℓ^1 appears once in positions i, j and k, and alternative c_ℓ^1 appears once in positions $i + 1$, $j + 1$ and $k + 1$. Then, the constraint (b_ℓ^1, c_ℓ^1) corresponds to the inequality $s_i - s_{i+1} + s_j - s_{j+1} + s_k - s_{k+1} > 0$ or, equivalently, $a_i + a_j + a_k > 0$.

- Alternative b_ℓ^2 appears once in position 1 and d/ϵ times in positions $i+1$, $j+1$ and $k+1$, and alternative c_ℓ^2 appears d/ϵ times in positions i, j and k. The constraint (b_ℓ^2, c_ℓ^2) corresponds to the inequality $s_1 - \frac{d}{\epsilon}s_i + \frac{d}{\epsilon}s_{i+1} - \frac{d}{\epsilon}s_j + \frac{d}{\epsilon}s_{j+1} - \frac{d}{\epsilon}s_k + \frac{d}{\epsilon}s_{k+1} > 0$ or, equivalently, $a_i + a_j + a_k < \epsilon$ (since $s_1 = d$, $a_i = s_i - s_{i+1}$, $a_j = s_j - s_{j+1}$ and $a_k = s_k - s_{k+1}$).

- Alternative b_ℓ^3 appears d times in positions i, j and k, and alternative c_ℓ^3 appears 2 times in position 1 and d times in positions $i+1$, $j+1$ and $k+1$. The constraint (b_ℓ^3, c_ℓ^3) corresponds to the inequality $-2s_1 + ds_i - ds_{i+1} + ds_j - ds_{j+1} + ds_k - ds_{k+1} > 0$ or, equivalently, $a_i + a_j + a_k > 2$.

- Alternative b_ℓ^4 appears $2/\epsilon + 1$ times in position 1 and d/ϵ times in positions $i+1$, $j+1$ and $k+1$, and alternative c_ℓ^4 appears d/ϵ times in positions i, j and k. The constraint (b_ℓ^4, c_ℓ^4) corresponds to the inequality $\left(\frac{2}{\epsilon} + 1\right) s_1 - \frac{d}{\epsilon}s_i + \frac{d}{\epsilon}s_{i+1} - \frac{d}{\epsilon}s_j + \frac{d}{\epsilon}s_{j+1} - \frac{d}{\epsilon}s_k + \frac{d}{\epsilon}s_{k+1} > 0$ or, equivalently, $a_i + a_j + a_k < 2 + \epsilon$.

In the case where equation ℓ is a 1-equation of the form $x_i \oplus x_j \oplus x_k = 1$, we have:

- Alternative b_ℓ^1 appears d times in positions i, j and k, and alternative c_ℓ^1 appears once in position 1 and d times in positions $i+1$, $j+1$ and $k+1$. Then, the constraint (b_ℓ^1, c_ℓ^1) corresponds to the inequality $-s_1 + ds_i - ds_{i+1} + ds_j - ds_{j+1} + ds_k - ds_{k+1} > 0$ or, equivalently, $a_i + a_j + a_k > 1$ (since $s_1 = d$, $a_i = s_i - s_{i+1}$, $a_j = s_j - s_{j+1}$ and $a_k = s_k - s_{k+1}$).

- Alternative b_ℓ^2 appears $1/\epsilon + 1$ times in position 1 and d/ϵ times in positions $i+1$, $j+1$ and $k+1$, and alternative c_ℓ^2 appears d/ϵ times in positions i, j and k. The constraint (b_ℓ^2, c_ℓ^2) corresponds to the inequality $\left(\frac{1}{\epsilon} + 1\right) s_1 - \frac{d}{\epsilon}s_i + \frac{d}{\epsilon}s_{i+1} - \frac{d}{\epsilon}s_j + \frac{d}{\epsilon}s_{j+1} - \frac{d}{\epsilon}s_k + \frac{d}{\epsilon}s_{k+1} > 0$ or, equivalently, $a_i + a_j + a_k < 1 + \epsilon$.

- Alternative b_ℓ^3 appears d times in positions i, j and k, and alternative c_ℓ^3 appears 3 times in position 1 and d times in positions $i+1$, $j+1$ and $k+1$. The constraint (b_ℓ^3, c_ℓ^3) corresponds to the inequality $-3s_1 + ds_i - ds_{i+1} + ds_j - ds_{j+1} + ds_k - ds_{k+1} > 0$ or, equivalently, $a_i + a_j + a_k > 3$.

- Alternative b_ℓ^4 appears $3/\epsilon + 1$ times in position 1 and d/ϵ times in positions $i+1$, $j+1$ and $k+1$, and alternative c_ℓ^4 appears d/ϵ times in positions i, j and k. The constraint (b_ℓ^4, c_ℓ^4) corresponds to the inequality $\left(\frac{3}{\epsilon} + 1\right) s_1 - \frac{d}{\epsilon}s_i + \frac{d}{\epsilon}s_{i+1} - \frac{d}{\epsilon}s_j + \frac{d}{\epsilon}s_{j+1} - \frac{d}{\epsilon}s_k + \frac{d}{\epsilon}s_{k+1} > 0$ or, equivalently, $a_i + a_j + a_k < 3 + \epsilon$.

In order for this profile to be valid, we use sufficiently many agents and additional alternatives (that do not appear in the constraints) as placeholders, so that the alternatives mentioned above have the appropriate number of appearances in the rankings.

Proof of correctness We now prove that there exists a variable assignment for the MAX-3LIN-2 instance that satisfies L of its equations if and only if there exists a scoring vector that satisfies constraints of total weight $11m + L$. As we have discussed above, this is enough to complete the proof.

Consider an assignment that satisfies L of the equations. Consider the scoring vector defined by setting $a_i = x_i + \epsilon/4$ for $i \in [d-1]$. Recall that we use $\epsilon \leq 1/d$ and, hence, $\sum_{i=1}^{d-1} a_i < d$; this is sufficient so that the corresponding scoring vector $\mathbf{s}$ has non-negative entries. This scoring vector satisfies:

- three out of the four inequalities corresponding to any variable x_i, since $a_i = \epsilon/4 \in (0, \epsilon)$ when $x_i = 0$ and $a_i = 1 + \epsilon/4 \in (1, 1 + \epsilon)$ when $x_i = 1$;
- three out of the four inequalities corresponding to any satisfied 0-equation $x_i \oplus x_j \oplus x_k = 0$ since $a_i + a_j + a_k = 3\epsilon/4 \in (0, \epsilon)$ when $x_i + x_j + x_k = 0$ and $a_i + a_j + a_k = 2 + 3\epsilon/4 \in (2, 2 + \epsilon)$ when $x_i + x_j + x_k = 2$;
- two out of the four inequalities corresponding to any unsatisfied 0-equation since $a_i + a_j + a_k \notin (0, \epsilon) \cup (2, 2 + \epsilon)$ in that case (observe that $a_i + a_j + a_k = 1 + 3\epsilon/4$ when $x_i + x_j + x_k = 1$ and $a_i + a_j + a_k = 3 + 3\epsilon/4$ when $x_i + x_j + x_k = 3$);
- three out of the four inequalities corresponding to any 1-equation $x_i \oplus x_j \oplus x_k = 1$ since $a_i + a_j + a_k = 1 + 3\epsilon/4 \in (1, 1 + \epsilon)$ when $x_i + x_j + x_k = 1$ and $a_i + a_j + a_k = 3 + 3\epsilon/4 \in (3, 3 + \epsilon)$ when $x_i + x_j + x_k = 3$;
- two out of the four inequalities corresponding to any unsatisfied 1-equation since $a_i + a_j + a_k \notin (1, 1 + \epsilon) \cup (3, 3 + \epsilon)$ then (again, observe that $a_i + a_j + a_k = 3\epsilon/4$ when $x_i + x_j + x_k = 0$ and $a_i + a_j + a_k = 2 + 3\epsilon/4$ when $x_i + x_j + x_k = 2$).

Hence, the total weight of the constraints satisfied is $3\sum_{i=1}^{n} m_i + 3L + 2(m - L) = 11m + L$, since $\sum_{i=1}^{n} m_i = 3m$ due to the fact that all equations have three variables and the sum $\sum_{i=1}^{n} m_i$ accounts for the total number of appearances of all variables in equations.

Conversely, assume that we are given a scoring vector that satisfies constraints of total weight $11m + L$; we will show that there exists an assignment to the variables of the MAX-3LIN-2 instance that satisfies L equations. First, we show that we can transform the scoring vector into a (possibly) different one with $a_i = \epsilon/4$ or $a_i = 1 + \epsilon/4$ for $i \in [d - 1]$, without decreasing the total weight of the satisfied constraints.

For a variable $a_i \notin (0, \epsilon) \cup (1, 1 + \epsilon)$ we have that the satisfied inequalities are the following: exactly two out of the four variable inequalities and at most three out of the four inequalities for each of the m_i equations in which the variable x_i appears. This gives a weight of at most $2m_i + 3m_i = 5m_i$. By setting $a_i = \epsilon/4$, exactly three out of the four variable inequalities and at least two out of the four equation inequalities in which x_i appears are satisfied, for a total weight of at least $5m_i$. Clearly, there is no loss in weight after this change in the value of a_i. So, in the following, we can assume that $a_i \in (0, \epsilon) \cup (1, 1 + \epsilon)$ for every variable a_i.

Now, we slightly modify the variable values as follows: for all variables $a_i \in (0, \epsilon)$ we set $a_i = \epsilon/4$ and for all variables $a_i \in (1, 1+\epsilon)$ we set $a_i = 1+\epsilon/4$. The set of inequalities containing a_i that were satisfied before the modification are still satisfied after the update as well. This is trivial for the variable inequalities. For $\beta \in \{0, 1, 2, 3\}$, for an equation inequality of the form $a_i + a_j + a_k < \beta + \epsilon$ (respectively, $a_i + a_j + a_k > \beta$) that was satisfied before the modification, at most β (respectively, at least β) of the three variables have values in $(1, 1 + \epsilon)$ before the modification. Clearly, the inequality is satisfied after the modification as well.

So, we can assume that we have total weight of $11m + L$ from satisfied constraints with the variables a_i taking values in $\{\epsilon/4, 1 + \epsilon/4\}$. Hence, $3\sum_{i=1}^{n} m_i = 9m$ comes as weight from satisfied variable inequalities (with three satisfied inequalities per variable). Then, the remaining weight comes from $2m + L$ satisfied equation inequalities. The definition of the reduction implies that there exist L equations in the MAX-3LIN-2 instance so that three among the four corresponding inequalities are satisfied. Then, it is easy to inspect that, if three among the four equation inequalities are satisfied when variables take values in $\{\epsilon/4, 1 + \epsilon/4\}$, then the (binary) assignment

	Sydney, Australia
	Oslo, Norway
	Baghdad, Iraq
	Vienna, Austria
	Washington, USA
	London, UK

	Greece
	Switzerland
	Nigeria
	Thailand
	China
	Mexico

Figure 2: An example of the sets from the ppl and col templates given to some agent. The particular format was used for building the real-world profiles. The blank column at the left is where the corresponding participant was required to define her ranking by putting distinct numbers from 1 to 6.

$x_i = a_i - \epsilon/4$ satisfies their corresponding equation as well. This yields an assignment with (at least) L satisfied equations and the proof is complete. □

6 Experiments

In this section, we present our experiments, which should be viewed as complementary to our theoretical work in the previous sections. We report on the execution of algorithms on two real-world OptPSR instances that we have generated as well as on numerous synthetic ones. In contrast to the theoretical analysis of the approximation algorithms in Section 4 which focuses on worst-case instances, here we are mainly interested in the average-case behavior of algorithms or scoring rules in scenarios that are close to ones that are likely to appear in practice. This explains the findings described in the following, according to which the observed performance of simple algorithms/scoring rules is much closer to optimality compared, for example, to the performance guarantees in Theorem 4.

Our experimental setup involves two different scenarios, to which we refer to as *ppl* and *col*. Each scenario is defined by a set of alternatives and by a *profile template* (or, simply, a *template*). The alternatives in the ppl scenario are the 48 countries that are listed in Table 6 of A. For the col scenario, the alternatives are the 36 cities that are listed in Table 7. The templates consist of equal-sized subsets of alternatives that each agent will be asked to rank. Specifically, the templates consist of 392 sets of six alternatives each. The distribution of the alternatives to the different sets of the ppl and col templates is almost uniform; each country appears in at least 47 and at most 52 sets, while each city appears in at least 57 and at most 70 sets. The templates are used to produce profiles as follows. Each profile has exactly 392 agents. Each agent is given a distinct set of alternatives from the template (see Figure 2 for examples of such sets for ppl and col) and ranks these alternatives. The profile then consists of the rankings provided by all agents.

Input profiles In our experiments, we used both real-world and synthetic data. Two real-world profiles (for the scenarios ppl and col, respectively) were collected as input from 392 participants[1] in the PatrasIQ[2] technology exhibition organized by our home institution in April 2016. Each

[1] Actually, we had prepared even more sets of alternatives that could be part of the template; as we did not manage to obtain more inputs, we restricted the templates and synthetic profiles to 392 agents for comparison reasons.

[2] `www.patrasiq.gr`

participant was given distinct sets of six countries and six cities (see Figure 2) from the ppl and col templates, and was asked to rank the countries in terms of their population and the cities in terms of their cost of living. These two profiles are available at `preflib.org` [32, 33] as dataset ED-00034.[3]

Many different synthetic profiles (in each scenario) were obtained by simulating agents who rank alternatives randomly. Specifically, each agent provides a noisy estimate of the correct underlying ranking of the alternatives assigned to her, according to either the Bradley-Terry [7] or the Plackett-Luce [29, 36] noise model. Both Bradley-Terry and Plackett-Luce are random utility models [2, 42]. They are defined using an underlying (positive) utility u_x associated with each alternative x and assume that the correct outcome of the pairwise relation between two alternatives x and y depends on the comparison between the utilities u_x and u_y (the alternative with the highest utility is better). In particular, the utilities of the 48 alternatives-countries in the ppl scenario are their populations according to information retrieved by `wikipedia.org` in April 2016 (see Table 6). Similarly, the utilities of the 36 alternatives-cities in the col scenario are their cost of living indices as retrieved by `numbeo.org` during the same time period (see Table 7).

A Bradley-Terry (BT, in short) agent (implicitly) works as follows. She first decides relations between all pairs of alternatives in her set. For each pair of alternatives x and y with corresponding utilities 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 in her set (that have been computed separately) define a ranking, then this is the ranking provided by the agent. Otherwise, the whole process is repeated from scratch.

A Plackett-Luce (PL, in short) agent decides the ranking of the alternatives in her set B sequentially. 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.

Constraints The constraints of the OptPSR instances we consider in our experiments were defined as follows. In both scenarios, we have a constraint (x, y) for each ordered pair of alternatives x and y such that $u_x > u_y$. For example, a constraint in the ppl scenario is the pair (China, Switzerland) as China is more populous than Switzerland. We consider three different weightings of the constraints, defining different OptPSR instances. In particular, the weight of (x, y) is either 1, or equal to $u_x - u_y$, or equal to $\log(u_x - u_y)$.

Unit weights are used when we care only about maximizing the number of correctly recovered pairwise comparisons between alternatives. However, there might be pairs that are really important to recover correctly, while some others are not. For example, in the ppl scenario, it might be important to conclude that China is ranked above Switzerland since their population difference is almost 1.3 billion people. Using this reasoning, an error in the comparison between Cuba and Belgium (both with population around 11 million) would not be that severe. Weighted and log-weighted (as opposed to unweighted) constraints have been introduced to capture this characteristic.

Evaluation Since all profiles that we experimented with have $d = 6$, one would expect that the exact algorithm `Regions` presented in Section 3 would be the obvious choice in order to come up with

[3] It seems that together with the dataset ED-00025 that were collected and used by Mao et al. [31], these are among the very few existing voting profiles with an underlying ground truth ranking.

the optimal scoring rule. Unfortunately, for the size of OptPSR instances that we considered (with $\binom{48}{2} = 1128$ constraints for ppl and $\binom{36}{2} = 630$ constraints for col), `Regions` (as well as algorithm `ApxPSR` from Section 4.2) turned out to be really slow, even after implementing several heuristics that yield minor performance improvements. 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 vector. Similarly, we have implemented a simplified variant of $\texttt{ApxPSR}_2$ by searching over all vectors of the form $(1, s, 0, 0, 0, 0)$, $(1, 1, 1, s, 0, 0)$, and $(1, 1, 1, 1, 1, s)$ where s is a multiple of 0.02 between 0 and 1. For real-world profiles, this approach has yielded the vectors that are shown in Table 4.

We remark that these variants of `Regions` and $\texttt{ApxPSR}_2$ are the most time-consuming among the algorithms we have implemented. The total execution time of all experiments (which were conducted using Matlab R2017B) is approximately 44 hours using an Intel 12-core i7 desktop, with 32Gb of RAM, running Windows 7. This is amortized to approximately 10 seconds per instance for the variant of `Regions` and less than 1 second per instance for the variant of $\texttt{ApxPSR}_2$.

rule	ppl	col
Optimal	$(1.00, 0.50, 0.35, 0.20, 0.15, 0.05)$	$(1.00, 0.90, 0.30, 0.30, 0.24, 0.00)$
$\texttt{ApxPSR}_2$	$(1.00, 0.02, 0.00, 0.00, 0.00, 0.00)$	$(1.00, 1.00, 1.00, 0.34, 0.00, 0.00)$

(a) Unweighted constraints

rule	ppl	col
Optimal	$(1.00, 0.65, 0.65, 0.35, 0.30, 0.25)$	$(1.00, 0.68, 0.68, 0.50, 0.22, 0.22)$
$\texttt{ApxPSR}_2$	$(1.00, 0.30, 0.00, 0.00, 0.00, 0.00)$	$(1.00, 1.00, 1.00, 0.52, 0.00, 0.00)$

(b) Weighted constraints

rule	ppl	col
Optimal	$(1.00, 0.60, 0.42, 0.20, 0.18, 0.08)$	$(1.00, 0.65, 0.65, 0.35, 0.30, 0.25)$
$\texttt{ApxPSR}_2$	$(1.00, 0.02, 0.00, 0.00, 0.00, 0.00)$	$(1.00, 1.00, 1.00, 0.52, 0.00, 0.00)$

(c) Log-weighted constraints

Table 4: The positional scoring vectors returned by the variants of the optimal algorithm and algorithm $\texttt{ApxPSR}_2$ on the real-world profiles. These vectors follow by searching a discretized spaces with scores that are multiples of either 0.05 or 0.02.

We compare the optimal OptPSR solution (obtained as described above) to the scoring vector returned by algorithm `BestApproval` which was presented in Section 4, the variant of $\texttt{ApxPSR}_2$ (also implemented as described above) and to two well-known scoring rules: the `Borda` count scoring rule that is defined by the scoring vector $(5, 4, 3, 2, 1, 0)$, as well as the `Harmonic` scoring rule (also known as Dowdall) which is defined by the vector $(1, 1/2, 1/3, 1/4, 1/5, 1/6)$. Table 5 shows the performance of these algorithms/scoring rules in all OptPSR instances that we experimented with. Each performance value is expressed as a percentage of the total weight of the constraints satisfied by an algorithm/scoring rule compared to the total weight of all constraints. The two columns labeled "real data" contain values that correspond to a single execution of an algorithm/scoring rule on the two real-world ppl and col profiles. The values in the remaining columns are averages over executions of algorithms/scoring rules on 1000 (random) profiles with either BT or PL agents as well as their standard deviations. Table 5 is split in three parts to report the results for OptPSR

instances with unweighted (Table 5a), weighted (Table 5b), and log-weighted constraints (Table 5c).

Our results indicate that `Harmonic` is better than both `Borda` and `BestApproval` in OptPSR instances with real-world profiles. `Harmonic` is also better than $\texttt{ApxPSR}_2$ in all real-world profiles besides ppl with weighted constraints. On the other hand, $\texttt{ApxPSR}_2$ is better the best algorithm for all synthetic profiles besides ppl with BT agents, where `Borda` is slightly better. Even though `BestApproval` is never the best among the four algorithms/scoring rules, it always provides competitive results. The performance values of algorithms/scoring rules on the real-world profiles are considerably inferior to those for synthetic profiles. This indicates that the BT and PL noise models are rather idealized and do not reflect accurately the behavior of the agents in our real-world inputs. Still, as they allowed for many executions, they made possible the assessment of the algorithms/scoring rules in terms of robustness, as we will see shortly.

Table 5b indicates that the performance values for weighted constraints are considerably higher than those for unweighted ones. This is to be expected since the total weight of correctly recovered constraints improves significantly when heavy pairwise relations are satisfied. The performance values for log-weighted constraints seem to lie in-between.

The low standard deviation values in Table 5 for synthetic profiles indicate that the performance values measured for each algorithm/scoring rule in the 1000 executions are always sharply concentrated around their average values. Indicatively, we demonstrate this by plotting a point for each execution, which has the performance value of the optimal scoring rule as x-coordinate and the performance value of `Borda` as the y-coordinate. The twelve 1000-point clouds in Figure 3 correspond to the three different constraint weightings and the profiles with BT and PL agents for the ppl and col scenarios. All points in these clouds lie below the red dashed diagonal as `Borda` is in general suboptimal. The clouds of points comparing the optimal scoring rule with the remaining algorithms/scoring rules (not reported here) have similar structure and, like Figure 3, suggest that the average values in Table 5 robustly characterize the performance of algorithms/scoring rules on all the synthetic instances that we have considered. Another interesting observation from these experiments is that the optimal scoring vectors corresponding to the different points of the clouds in Figure 3 are, in general, very different, in spite of the fact that the optimal performance values are so concentrated.

7 Conclusions and open problems

Motivated by crowdsourcing and rating applications, we have introduced and studied the OptPSR problem. Very informally, the problem is to compute a positional scoring rule, whose outcome when applied on a given profile is as close as possible to constraints corresponding to underlying correct pairwise relations between alternatives. We have presented the algorithm `Regions` that solves the problem exactly by cleverly searching the space of all candidate scoring vectors and exploiting linear programming. The algorithm runs in polynomial time when the parameter d (representing the number of alternatives in the ranking of each agent) is constant. We also consider approximation algorithms for OptPSR. A simple algorithm, called `BestApproval`, that selects among all approval vectors the one that satisfies constraints of the highest weight is shown to achieve a tight approximation ratio of $1/d$. Our more sophisticated algorithm `ApxPSR` can achieve better approximation ratios at the expense of higher running times. We complement these positive results by showing that OptPSR is a hard-to-approximate optimization problem. We also present an experimental evaluation of algorithms and scoring rules on real-world and synthetic instances.

	real data		synthetic (BT)				synthetic (PL)			
	ppl	col	ppl		col		ppl		col	
rule			avg	std	avg	std	avg	std	avg	std
Optimal	81.83	83.97	94.54	0.642	93.74	1.037	93.19	0.916	88.20	1.867
`ApxPSR`$_2$	80.50	82.22	93.22	0.774	**92.74**	1.019	**92.04**	0.963	**86.68**	1.941
`Borda`	79.96	82.06	**93.43**	0.706	92.04	1.112	91.94	0.983	85.95	1.984
`Harmonic`	**80.94**	**82.54**	92.85	0.746	91.35	1.297	90.50	1.109	83.18	2.321
`BestApproval`	79.43	80.48	91.72	0.844	91.42	1.080	90.54	0.986	84.68	1.929

(a) Unweighted constraints

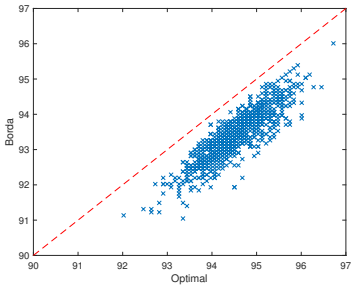
	real data		synthetic (BT)				synthetic (PL)			
	ppl	col	ppl		col		ppl		col	
rule			avg	std	avg	std	avg	std	avg	std
Optimal	95.98	92.93	99.66	0.078	98.69	0.364	99.49	0.134	96.02	1.079
`ApxPSR`$_2$	**95.62**	91.95	99.47	0.117	**98.34**	0.412	**99.31**	0.172	**95.13**	1.236
`Borda`	94.67	91.92	**99.52**	0.102	98.07	0.447	99.30	0.166	94.82	1.260
`Harmonic`	95.42	**92.01**	99.42	0.124	97.80	0.542	99.04	0.229	92.67	1.769
`BestApproval`	95.24	90.83	99.34	0.143	97.90	0.472	99.10	0.196	94.18	1.313

(b) Weighted constraints

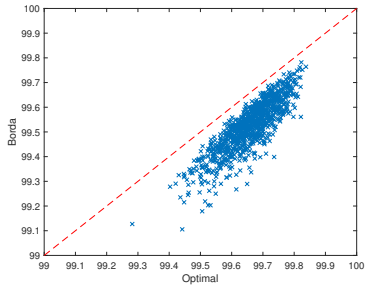
	real data		synthetic (BT)				synthetic (PL)			
	ppl	col	ppl		col		ppl		col	
rule			avg	std	avg	std	avg	std	avg	std
Optimal	83.03	88.21	95.29	0.553	97.02	0.717	94.06	0.768	92.53	1.487
`ApxPSR`$_2$	81.56	86.82	94.07	0.681	**96.24**	0.765	**93.06**	0.887	**91.24**	1.695
`Borda`	81.05	86.91	**94.31**	0.596	95.87	0.841	92.94	0.827	90.75	1.663
`Harmonic`	**82.15**	**87.24**	93.75	0.674	95.32	0.952	91.59	1.015	88.08	2.076
`BestApproval`	80.56	85.48	92.79	0.722	95.52	0.841	91.65	0.872	89.77	1.633

(c) Log-weighted constraints

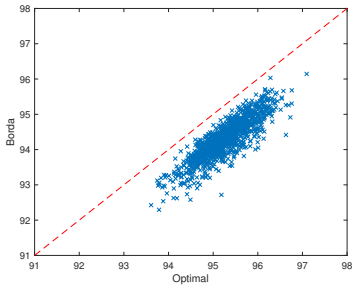
Table 5: Performance of algorithms/scoring rules on instances with weighted constraints. Each value is a percentage and denotes the ratio of the total weight of the satisfied constraints over the total weight of all constraints. For synthetic profiles with BT and PL agents, the values indicate average performance (avg) and standard deviations (std) from 1000 simulations (on randomly generated BT and PL profiles), while for real-world data each value corresponds to a single execution of an algorithm/scoring rule. The performance of the best among the non-optimal scoring rules is marked in bold. As the ppl and col instances have 1128 and 630 constraints, a respective difference of 0.09 and 0.16 in performance in subtable (a) corresponds to a gain difference of one constraint.



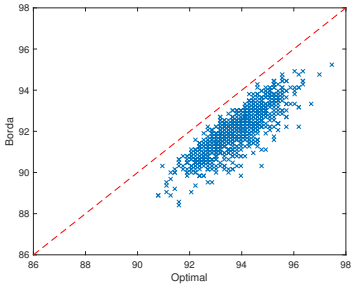
(a) BT, ppl, unweighted



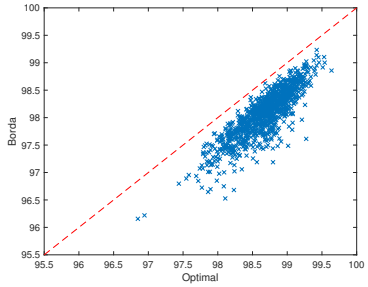
(b) BT, ppl, weighted



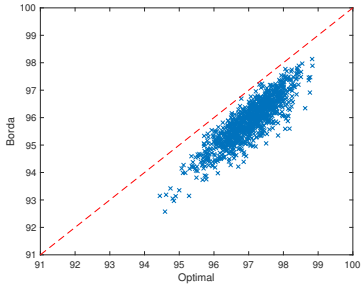
(c) BT, ppl, log-weighted



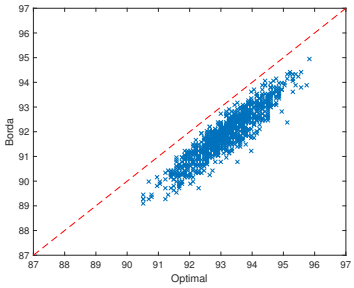
(d) BT, col, unweighted



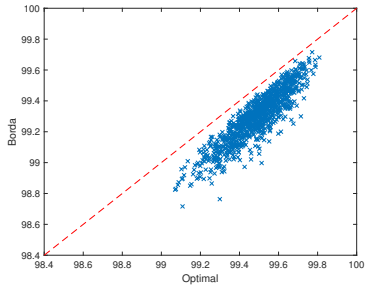
(e) BT, col, weighted



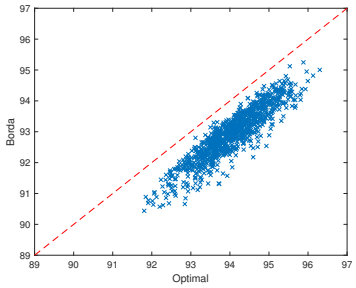
(f) BT, col, log-weighted



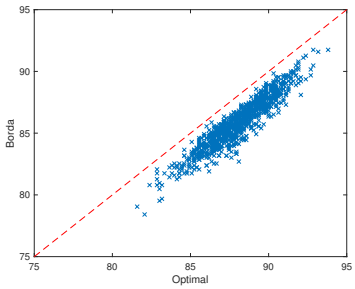
(g) PL, ppl, unweighted



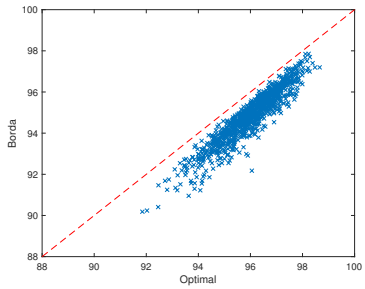
(h) PL, ppl, weighted



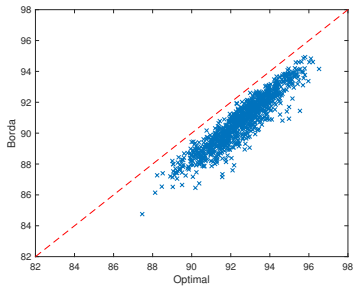
(i) PL, ppl, log-weighted



(j) PL, col, unweighted



(k) PL, col, weighted



(l) PL, col, log-weighted

Figure 3: Comparing `Borda` with the optimal OptPSR solution. Each cloud consists of 1000 points, each corresponding to a distinct simulation. The x-coordinate of each point is the performance value of the optimal scoring rule and the y-coordinate is the performance value of `Borda`. The caption of each subfigure indicated the type of agents (BT or PL), the scenario (ppl or col), and the constraint weighting (unweighted, weighted, or log-weighted).

Our work reveals several open problems. First, we would like to design exact algorithms that are practical. Our ambitious goal here is to be able to solve —in reasonable time— OptPSR instances like the ones we used in our experiments (i.e., with d up to 10, approximately 50 alternatives, and 1000 constraints).

Second, we would like to determine the approximability of OptPSR. Currently, there is a huge gap between our positive algorithmic results in Theorems 4 and 6 and the inapproximability bound from Theorem 7; the former have a dependence on d while the latter is a constant close to 1. Is there a polynomial time algorithm with constant approximation ratio? Is there a sub-constant inapproximability bound? These questions are very important from the theoretical point of view. More importantly, we would like to design practical approximation algorithms that will be effective on huge OptPSR instances. Here, we need both simplicity and efficiency; achieving these two goals simultaneously seems elusive at this point.

Third, observe that our theoretical results in Section 4 focus on worst-case approximation guarantees. In addition to such studies, we would like to conduct theoretical analysis in random profiles that have been produced by Plackett-Luce or Bradley-Terry agents or, more importantly, by more realistic agents who follow appropriate generalizations of random utility models (see [2]). In particular, the following optimization problem that is inspired by the flavor of our experiments is very appealing: Given a template, constraints, and statistical information (e.g., a noise model) describing the behavior of agents, compute the best algorithm or scoring rule that maximizes the expected total weight of satisfied constraints. Here, the expectation is taken over random OptPSR instances with agents following the given noise model that are asked to rank the sets of alternatives in the template.

Finally, our definition of OptPSR assumes that all agents rank the same number of alternatives. This feature has been used for proof-of-concept purposes here but, admittedly, it could be very restrictive in many applications. Extending OptPSR by allowing different numbers of alternatives per agent (and more general definitions of scoring vectors) is important.

Thinking beyond OptPSR, one could consider optimization problems of similar flavor by replacing positional scoring rules by a class of voting rules defined over incomplete votes which can be identified by a small number of parameters (in the same way in which positional scoring rules are identified by the position scores). One possibility might be to consider the class of Kemeny-like voting rules which given a profile of possibly incomplete votes computes a full ranking of the alternatives that has the minimum possible total distance from the votes of the profile. Each voting rule in this class is identified by a distance function between rankings. Kemeny is such a voting rule for the Kendall-tau distance function (see [47]). Then, a natural optimization problem would aim for optimizing the distance function parameters so that the resulting voting rule, when applied on the given profile, returns a ranking that is as close as possible to the constraints of an underlying true ranking. This direction might be worth studying, taking extra care of computational issues (e.g., Kemeny is computationally hard to resolve) that do not arise in OptPSR.

References

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

[2] H. Azari Soufiani, D. C. Parkes, and L. Xia. Random utility theory for social choice. In *Proceeding of the 26th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 126–134, 2012.

[3] H. Aziz, O. Lev, N. Mattei, J. S. Rosenschein, and T. Walsh. Strategyproof peer selection: Mechanisms, analyses, and experiments. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, pages 397–403, 2016.

[4] Dorothea Baumeister, Piotr Faliszewski, Jérôme Lang, and Jörg Rothe. Campaigns for lazy voters: truncated ballots. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 577–584, 2012.

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

[6] Craig Boutilier and Jeffrey S. Rosenschein. Incomplete information and communication in voting. In *Handbook of Computational Social Choice*, pages 223–258. 2016.

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

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

[9] Mark Braverman and Elchanan Mossel. Noisy sorting without resampling. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 268–276, 2008.

[10] I. Caragiannis, G. A. Krimpas, and A. A. Voudouris. 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)*, pages 675–683, 2015.

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

[12] I. Caragiannis, A. D. Procaccia, and N. Shah. When do noisy votes reveal the truth? *ACM Transactions on Economics and Computation*, 4(3):15:1–15:30, 2016.

[13] Flavio Chierichetti and Jon M. Kleinberg. Voting with limited information and many alternatives. *SIAM Journal of Computing*, 43(5):1615–1653, 2014.

[14] V. Conitzer and T. Sandholm. Common voting rules as maximum likelihood estimators. In *Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence (UAI)*, pages 145–152, 2005.

[15] Marquis de Condorcet. Essai sur l'application de l'analyse à la probabilité de décisions rendues à la pluralité de voix. Imprimerie Royal, 1785. Facsimile published in 1972 by Chelsea Publishing Company, New York.

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

[17] John A. Doucette, Kate Larson, and Robin Cohen. Conventional machine learning for social choice. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 858–864, 2015.

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

[19] Edith Elkind and Arkadii Slinko. Rationalizations of voting rules. In *Handbook of Computational Social Choice*, pages 169–196. 2016.

[20] Y. Filmus and J. Oren. Efficient voting via the top-k elicitation scheme: A probabilistic approach. In *Proceedings of the 15th ACM conference on Economics and Computation (EC)*, pages 295–312, 2014.

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

[22] G. A. Hazelrigg. Dear colleague letter: Information to principal investigators (PIs) planning to submit proposals to the Sensors and Sensing Systems (SSS) program October 1, 2013, deadline. NSF website, 2013.

[23] K. Konczak and Jérôme Lang. Voting procedures with incomplete preferences. In *Proceedings of the Multidisciplinary IJCAI-05 Workshop on Advances in Preference Handling*, pages 124–129, 2005.

[24] David Kurokawa, Omer Lev, Jamie Morgenstern, and Ariel D. Procaccia. Impartial peer review. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 582–588, 2015.

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

[26] T. Lu and C. Boutilier. Effective sampling and learning for mallows models with pairwise-preference data. *Journal of Machine Learning Research*, 15:3963–4009, 2014.

[27] Tyler Lu and Craig Boutilier. Robust approximation and incremental elicitation in voting protocols. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 287–293, 2011.

[28] Tyler Lu and Craig Boutilier. Multi-winner social choice with incomplete preferences. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 263–270, 2013.

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

[30] C. L. Mallows. Non-null ranking models. *Biometrika*, 44:114–130, 1957.

[31] A. Mao, A. D. Procaccia, and Y. Chen. Better human computation through principled voting. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI)*, pages 1142–1148, 2013.

[32] N. Mattei and T. Walsh. Preflib: A library for preferences http://www.preflib.org. In *Proceedings of the 3rd International Conference on Algorithmic Decision Theory (ADT)*, pages 259–270, 2013.

[33] Nicholas Mattei and Toby Walsh. A PrefLib.org retrospective: Lessons learned and new directions. In Ulle Endriss, editor, *Trends in Computational Social Choice*, chapter 15, pages 289–305. AI Access, 2017.

[34] Michael R. Merrifield and Donald G. Saari. Telescope time without tears: a distributed approach to peer review. *Astronomy and Geophysics*, 50(4):4.2–4.6, 2009.

[35] Maria Silvia Pini, Francesca Rossi, Kristen Brent Venable, and Toby Walsh. Incompleteness and incomparability in preference aggregation: Complexity results. *Artificial Intelligence*, 175(7-8):1272–1289, 2011.

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

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

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

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

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

[41] N. B. Shah and M. J. Wainwright. Simple, robust and optimal ranking from pairwise comparisons. *Journal of Machine Learning Research*, 18:199:1–199:38, 2017.

[42] L. L. Thurstone. A law of comparative judgement. *Psychological Review*, 34(4):273–286, 1927.

[43] H.E. Warren. Lower bounds for approximation by non-linear manifolds. *Transaction of the American Mathematical Society*, 133:167–178, 1968.

[44] L. Xia and V. Conitzer. A maximum likelihood approach towards aggregating partial orders. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 446–451, 2011.

[45] Lirong Xia and Vincent Conitzer. Determining possible and necessary winners given partial orders. *Journal of Artificial Intelligence Research*, 41:25–67, 2011.

[46] H. P. Young. Condorcet's theory of voting. *The American Political Science Review*, 82(4):1231–1244, 1988.

[47] William S. Zwicker. Introduction to the theory of voting. In *Handbook of Computational Social Choice*, pages 23–56. 2016.

A Alternatives and utilities used in experiments

Here we present the alternatives and the utilities that define the correct underlying ranking, the constraints, and the corresponding weightings in our experiments. Table 6 contains the list of 48 countries and their populations sorted in descending order, as retrieved from `wikipedia` in April 2016 (ppl scenario). Table 7 contains the list of 36 cities and their corresponding cost of living index sorted in decreasing order, as retrieved from the website `numbeo.com` in April 2016 (col scenario).

countries	population	countries	population	countries	population
China	1,375,880,000	Iran	79,149,100	Peru	31,488,700
India	1,287,180,000	Turkey	78,741,053	Australia	24,051,600
USA	323,225,000	Thailand	65,273,832	Romania	19,861,000
Indonesia	258,705,000	Great Britain	65,097,000	Chile	18,191,900
Brazil	205,900,000	France	64,543,000	Netherlands	17,003,600
Pakistan	193,295,802	Italy	60,676,361	Belgium	11,312,444
Nigeria	186,988,000	South Korea	51,569,536	Cuba	11,238,317
Bangladesh	160,197,000	Colombia	48,608,000	Greece	10,864,979
Russia	146,544,710	Kenya	47,251,000	Czech Republic	10,553,843
Japan	126,920,000	Spain	46,423,064	Portugal	10,374,822
Mexico	122,273,500	Argentina	43,590,400	Sweden	9,866,670
Philippines	103,083,100	Ukraine	42,738,070	Hungary	9,849,000
Ethiopia	92,206,005	Algeria	40,400,000	Austria	8,699,730
Vietnam	91,700,000	Iraq	36,575,000	Israel	8,489,400
Egypt	90,755,700	Canada	36,048,521	Switzerland	8,306,200
Germany	81,459,000	Saudi Arabia	32,248,200	Bulgaria	7,202,198

Table 6: The 48 countries that are used as alternatives in the ppl scenario, ordered by population. Retrieved from `wikipedia.org` (April 2016).

cities	col index	cities	col index	cities	col index
San Francisco	111.67	Doha	68.99	Barcelona	47.91
Zurich	106.19	Stockholm	66.46	Montreal	46.87
New York	100.00	Melbourne	62.25	Lagos	44.19
Lausanne	93.72	Tel Aviv	61.89	Nicosia	41.58
London	89.23	Munich	59.72	Athens	37.19
Washington	85.67	Rome	58.93	Istanbul	34.74
Boston	80.86	Brussels	58.29	Baghdad	33.42
Oslo	76.31	Toronto	55.75	Patras	32.21
Sydney	75.92	Maastricht	54.70	Budapest	30.69
Tokyo	74.35	Vienna	52.59	City of Mexico	29.20
Dubai	73.07	Genoa	51.11	Bucharest	27.10
Copenhagen	71.12	Berlin	48.96	Mumbai	24.64

Table 7: The 36 cities that are used as alternatives in the col scenario, ordered by cost of living (plus rent) index. Retrieved from `numbeo.com` (April 2016).