Reducing Bias in Preference Aggregation for Multiagent Soft Constraint Problems

Alexander Schiendorfer(✉) and Wolfgang Reif

Institute for Software & Systems Engineering, University of Augsburg, Augsburg, Germany
{schiendorfer,reif}@isse.de

Abstract. Most distributed constraint optimization problems assume the overall objective function to be the "utilitarian social welfare", i.e., a sum of several utility functions, belonging to different agents. This also holds for the most popular soft constraint formalisms, cost function networks and weighted constraints. While, in theory, this model is sound, it is susceptible to manipulation and resulting bias in practice. Even without malevolent intentions, bias can result from the way orderings over solutions are transformed into numerical values or normalized. Alternatively, preferences can be aggregated directly using the tools of social choice theory to discourage manipulations and practically reduce unwanted bias. Several common voting functions can be implemented on top of constraint modeling languages through incremental search and suitable improvement predicates. We demonstrate that our approach, in particular Condorcet voting, can undo bias which is shown on two real-life-inspired case studies using the soft constraint extension MiniBrass on top of MiniZinc.

Keywords: Soft constraints · Distributed constraint optimization · Social choice · Modeling languages · MiniZinc

1 Motivation

Many real-life problems such as coordinating a fleet of mobile sensors [31] or scheduling devices in smart grids and homes [13] have recently been reduced to distributed constraint optimization problems (DCOP) that involve multiple agents. Similarly, more mundane tasks such as assigning seminar topics or agreeing on a shared meal plan are problems that (logically) involve several agents, even if solved centrally. By far the most popular way of aggregating agents' preferences (including the cited examples) is to assume them to be specified as (numerical) cost/utility functions that are summed up [12]. This is equivalent to weighted constraint satisfaction problems (WCSP), the most common class of soft constraint problems [18]. While this may be tolerable if other measures constrain the utilities, this commonly accepted notion of "social welfare" is prone to unfairness and bias in practice, especially if the utilities are unconstrained

T. Schiex and S. de Givry (Eds.): CP 2019, LNCS 11802, pp. 510–526, 2019.
https://doi.org/10.1007/978-3-030-30048-7_30

$$X = \{f, s\}, \qquad D_f = D_s = \{\text{curry}, \text{chili}, \text{stew}\}$$
$$\text{sols(CSP)} = \{(f \mapsto \text{stew}, s \mapsto \text{stew}), (f \mapsto \text{stew}, s \mapsto \text{curry}), (f \mapsto \text{curry}, s \mapsto \text{chili})\}$$

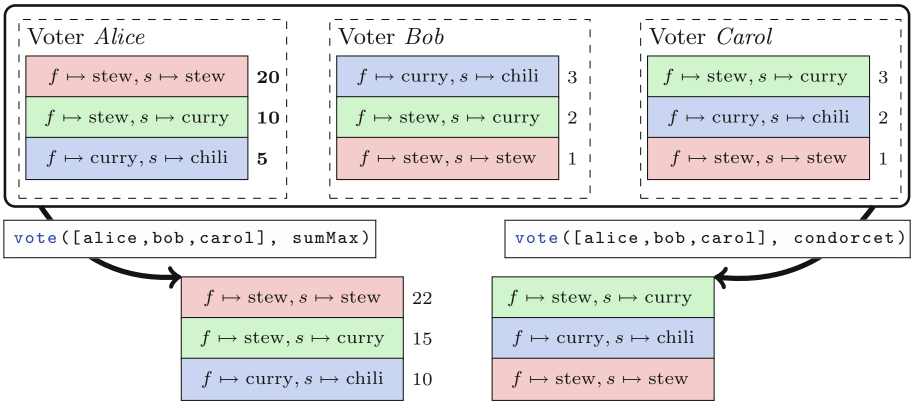

Voter *Alice*
$f \mapsto$ stew, $s \mapsto$ stew **20**
$f \mapsto$ stew, $s \mapsto$ curry **10**
$f \mapsto$ curry, $s \mapsto$ chili **5**
Voter *Bob*
$f \mapsto$ curry, $s \mapsto$ chili 3
$f \mapsto$ stew, $s \mapsto$ curry 2
$f \mapsto$ stew, $s \mapsto$ stew 1
Voter *Carol*
$f \mapsto$ stew, $s \mapsto$ curry 3
$f \mapsto$ curry, $s \mapsto$ chili 2
$f \mapsto$ stew, $s \mapsto$ stew 1
vote([alice,bob,carol], sumMax)
vote([alice,bob,carol], condorcet)
$f \mapsto$ stew, $s \mapsto$ stew 22
$f \mapsto$ stew, $s \mapsto$ curry 15
$f \mapsto$ curry, $s \mapsto$ chili 10
$f \mapsto$ stew, $s \mapsto$ curry
$f \mapsto$ curry, $s \mapsto$ chili
$f \mapsto$ stew, $s \mapsto$ stew

Fig. 1. Social welfare functions can reduce weight-induced bias. Agents pick meals for Friday f and Saturday s, with three solutions in sols(CSP) due to hard constraints. *Alice* submits manipulated weights which puts $(f \mapsto \text{stew}, s \mapsto \text{stew})$ to the top whereas the other agents like it least. Condorcet voting mitigates that by ranking $(f \mapsto \text{stew}, s \mapsto \text{curry})$ first since it wins both pairwise majority contests against the other solutions.

and only known at runtime (agents can just outbid each other). The left part (`sumMax`) of Fig. 1 illustrates this rather naïvely by allowing an agent *Alice* to vote with higher weights and manipulate the group decision in her favor.

There are, broadly speaking, three approaches to the problem:

(i) We normalize a single objective function to be less biased.
(ii) We address the problem as a multi-objective optimization problem looking for the Pareto front.
(iii) We devise *ordinal* operators that operate on the preference relations.

Regarding (i), of course, we would try to take care of such blunt manipulations as those in Fig. 1 in real-life problems. There are still more subtle ways how biased weights can emerge in preference specifications: Assume, e.g., that students rank seminar topics. Unless we make every student rank *every* topic, we are forced to introduce bias: consider student A stating six preferences whereas student B only specifies three. How should we relate a violation of A's top choice to one of B's top preference? The only fact we can safely deduce is which outcomes A and B prefer in isolation. Naïvely modeled as WCSPs, A's top choice is weighted six whereas B's top choice gets a weight of three. Summing them up clearly favors solutions that please A. Alternatively, we could allocate a fixed budget of q to every agent that proportionally distributes q. For instance, A could split 21 points as $\langle 6, 5, 4, 3, 2, 1\rangle$ whereas B could split the same 21 points as $\langle 8, 7, 6 \rangle$.[1] A solver then caters to B since A has more options sharing the fixed budget. Either way, the model is biased, independent of the subsequent solvers.

Conversely, following (ii), for multi-objective optimization, we consider each voters' weights as individual objective functions and calculate the Pareto front which contains all solutions that are not Pareto-dominated, i.e., not dominated by another solution in *all* dimensions [11]. Unfortunately, this concept alone is too weak (i.e., indecisive) for multi-agent problems. As the number of voters increases, any solution is more likely to be in the Pareto front, as a crude estimation of the ratio of Pareto-optimal solutions illustrates: Assume that n voters pick their top-preference out of m options at random. It suffices for a solution to be Pareto-optimal if at least one of the agents prefers it most. For a single agent, a solution θ stands a $\frac{m-1}{m}$ chance of *not being* top. The probability that *all* voters do not rank θ as top is thus $(\frac{m-1}{m})^n$ which immediately leads to the probability of θ being top for *at least one* voter: $1 - (\frac{m-1}{m})^n$. If a problem had $m = 100$ possible solutions and $n = 5$ voters, each solution would have a 4.9% chance of being Pareto-optimal. Raising the number of voters to 20 increases this chance to 18.2%, and with 40 voters, every solution already has a 33.1% chance of ending up Pareto-optimal. When facing such a large number of Pareto-optimal solutions, our problem is to be more selective within the Pareto-front – we would still insist on *at least* choosing a solution within the Pareto front since otherwise *all* agents agree that another one is better for them.

In terms of purely ordinal operators (alternative iii), Pareto and lexicographic combinations are the canonical combinations of preference relations [1] that do not need numeric utilities. Yet, we already discussed shortcomings of Pareto orderings and the lexicographic combination is a too strict form of preference aggregation. This is where *social choice theory* [3] comes into play. Rooted in electoral systems[2], it discusses how to amalgamate a group's preference relations. On the right side of Fig. 1, we see that voting based on only ordinal information (here, Condorcet's method that prefers an option to another if a majority favors it) can lead to fairer decisions. Voting over *solutions* to a constraint satisfaction problem corresponds to traversing the search space effectively (e.g., by constraint propagation and search heuristics). Therefore, we implement our approach with modeling languages that compile to a variety of algorithmically efficient solvers.

Our contribution in this paper is thus **to make voting methods such as Condorcet's amenable to soft constraint optimization on the modeling language level**. We extend the open source soft constraint modeling language MiniBrass [24] built on top of MiniZinc [20]. In contrast to other approaches (see Sect. 2.3) this allows agents to vote on the *solution level* instead of the individual *variable level*. Our key insight is that some voting methods can be conveniently mapped to branch-and-bound search.

[1] The same logic obviously applies to the normalized case of $q = 1$ where, e.g., B would get $[1/2, 1/3, 1/6]$ and A's top choice only gets a weight of 0.28.

[2] For instance, the Schulze method [27] is a Condorcet-based method used for elections in open source organizations such as Ubuntu, Debian, or the Wikimedia Foundation.

2 Preliminaries

Our approach combines (soft) constraint programming and social choice theory. In essence, soft constraint programming orders solutions using an overall valuation (not necessarily numerical) in a (partially) ordered set that results from individual valuations using a combination operation in an algebraic structure. Social choice theory deals with aggregating preference relations over outcomes to a single relation. Hence, these two ideas naturally complement each other.

2.1 Constraint Optimization and Soft Constraints

As usual, a *constraint (satisfaction) problem* CSP $= (X, D, C)$ is described by a set of (decision) variables X, their associated family of domains $D = (D_x)_{x \in X}$ of possible values, and a set of (hard) constraints C that restrict valid assignments. For a CSP, an assignment θ over scope X is a function from X to D such that each variable x maps to a value in D_x. The set of all assignments is written as $[X \to D]$. A (hard) constraint $c \in C$ is a function $c : [X \to D] \to \mathbb{B}$ where $\theta \models c$ expresses that θ satisfies c. For solving by inference, i.e., reducing valid domain items by logical implications (so-called *constraint propagation*), *global constraints* offer dedicated filtering algorithms [4]. Consequently, an assignment θ is a solution if $\theta \models c$ holds for all $c \in C$. We write $\theta \in \mathrm{sols}(\mathrm{CSP})$.

We move from satisfaction to *constraint optimization problems* (COP) by adding an objective function $f : [X \to D] \to P$ where $(P, \leq_P)$ is a partial order, i.e., $\leq_P$ is a reflexive, antisymmetric, and transitive relation over P. Elements of P are called *satisfaction degrees*. Without loss of generality, we interpret $m <_P n$ as satisfaction degree m being strictly *worse* than n and restrict our attention to *maximization problems* regarding P. Consequently, θ_1 is worse than θ_2 if $f(\theta_1) <_P f(\theta_2)$ which results in a partial quasi-ordering over solutions since multiple solutions may map to the same satisfaction degree and anti-symmetry does not hold. A solution θ is *optimal* with respect to a COP if for all solutions θ' it holds either that $f(\theta') \leq_P f(\theta)$ or $f(\theta') \parallel_P f(\theta)$, expressing incomparability with respect to $\leq_P$.

Soft constraint problems are specialized COPs where each soft constraint s_i maps $[X \to D]$ to an algebraic structure $(M, \cdot_M, \varepsilon_M, \leq_M)$, i.e., a partially-ordered, commutative monoid called a *partial valuation structure* (PVS) [15] which subsumes several specific soft constraint formalisms such as WCSP, cost function networks, or fuzzy constraints [18]. The combination operator $\cdot_M$ aggregates all soft constraints' satisfaction degrees, ε_M denotes maximal satisfaction and is neutral with respect to the $\cdot_M$ operator. In terms of COPs, the overall objective $f : [X \to D] \to (M, \leq_M)$ is defined by $f(\theta) = \Pi_M\{s_i(\theta) \mid s_i \in S\}$ for a set of soft constraints S, also written as $S(\theta)$. We use PVS for soft constraint problems since they are more general than c-semirings [5] or total valuation structures [25]. Therefore, they are used as basic building block in MiniBrass [24]. Since in this paper we only care about aggregating several agents' overall satisfaction degrees, we do not rely on the precise properties of the algebraic structure that are described in [15,24]. Given two PVS M and N, the most natural combination is the Cartesian product $M \times N$ that orders elements according to a Pareto ordering:

$$(m, n) \leq_{M \times N} (m', n') \Leftrightarrow m \leq_M m' \wedge n \leq_N n'$$

The Pareto-ordering leads to a "fair" but not decisive aggregation of several PVS, as we discussed in the introductory section.
By contrast, the ordering of the lexicographic combination is defined as

$$(m, n) \leq_{M \ltimes N} (m', n') \Leftrightarrow (m <_M m') \vee (m = m' \wedge n \leq_N n')$$

It allows us to express strictly hierarchical relationships between PVS to distinguish, e.g., organizational from individual goals.

In terms of software implementations, most existing constraint solvers offer an API to model constraint problems in imperative code. For higher layers of abstraction, there have been several proposals for domain-specific languages, including MiniZinc [20] or Essence [14]. Due to its popularity (see, e.g., the annual MiniZinc challenge [29]) and language features, we favor the former in this paper. MiniZinc is a high-level constraint modeling language understood by many constraint, MIP, or SAT solvers:

```
array[1..3] of var 1..3: x;                          % decision variables
constraint forall (i in 1..2) (x[i] <= x[i+1]);      % constraints
solve satisfy; % minimize sum(x) / maximize sum(x)   % objectives
```

In its default version, MiniZinc allows for a rich variety of global constraints but only limited capacity for optimization objectives. Only totally ordered numeric objectives are supported. To increase generality, incremental search is needed that facilitates adding and retracting new constraints during traversal of the search tree. MiniSearch [22] enables such customizable search on the solution level and MiniZinc itself has extensions for large neighborhood search where some variables are fixed to stay unchanged (using added constraints), thereby defining a large neighborhood [10]. MiniBrass [24] "softens" MiniZinc to incorporate PVS-based soft constraints. Agents specify their preferences as PVS and aggregate them using lexicographic or Pareto combinations, as Fig. 2 shows.

2.2 Social Choice Theory

Formally, the field of social choice theory is concerned with aggregating preference relations [3]. For a (usually finite and small) set of outcomes (or candidates) O, we call $\mathcal{Q}_O$, $\mathcal{P}_O$, and $\mathcal{T}_O$ the sets of quasi, partial, and total orders over O, respectively. Most often in social choice, quasi-orders (total, transitive but allowing for ties at the cost of anti-symmetry) are used whereas SCSPs lead to partial quasi-orders over solutions. We denote a set of agents (or voters) as $N = \{1, \ldots, n\}$. Then, a *preference profile* $[\preceq] = (\preceq_i)_{i \in N} \in \mathcal{P}_O^n$ (or $\mathcal{T}_O^n$, etc.) is a tuple containing a preference relation $\preceq_i$ for every agent $i \in N$ where, again, $o \preceq_i o'$ indicates that outcome o is *worse* than o'. Voting methods then map

```
PVS: alice = new
 WeightedCsp("alice") {
  scons c1:'f = stew /\ s = stew' :: w('20');
  scons c2:'f = stew /\ s = curry':: w('10');
  scons c3:'f = curry /\ s = chili':: w('5');
};
% PVS: carol = new FuzzyCsp("carol") {
```

```
PVS: bob = new
  ConstraintPreferences("bob") {
    scons c1: 'f = curry /\ s = chili';
    scons c2: 'f = stew /\ s = curry';
    scons c3: 'f = stew /\ s = stew';
    crEdges : '[| mbr.c3, mbr.c2 |
                  mbr.c2, mbr.c1 |]';
};
```

```
solve alice lex (bob pareto carol);
```

Fig. 2. Example problem in MiniBrass, slightly adapted from Fig. 1. Constraint preferences require ordinal information only (`mbr.c3, mbr.c2` denotes that $\mathtt{c}_3$ is less important than $\mathtt{c}_2$. Other PVS types (fuzzy) could be used as well.

a preference profile either to a single winning outcome or to a full preference relation over O. Both tasks strive to represent the agents' joint wishes. A *social welfare function* W maps a preference profile $[\preceq]$ of n agents to a preference relation, formally written as $W : \mathcal{P}_O^n \rightarrow \mathcal{P}_O$. By contrast, a *social choice function* $C : \mathcal{P}_O^n \rightarrow O$ only returns a winner [28]. The problems are strongly related since we convert a social welfare function to a social choice function by picking a top option, or conversely, repeatedly call a social choice function to obtain a full ordering as a social welfare function.

To list a few examples, the *majority voting* rule builds a welfare ordering by ranking all outcomes according to the number of top occurrences they achieve. *Borda voting* asks every agent to assign a score from 0 (least desirable) to $|O|-1$ (most desirable) and adds up the scores of all agents. *Condorcet voting* fixes an ordering over O, say $[o_1, \dots, o_m]$, and performs pairwise competitions to determine the welfare ordering. That is, agents vote for o_1 or o_2 and the winner (according to a majority) challenges o_3, and so on. A *Condorcet winner* is an outcome that wins all pairwise competitions. There can however be cycles such that no proper ordering emerges (e.g., transitivity is violated) [3]. Approval voting, on the other hand, lets agents only partition the set of outcomes into "approved" and "disapproved" and ranks outcomes by their number of approvals.

Besides these examples of voting methods, social choice theory offers several general impossibility results based on axiomatic characterizations of welfare functions, most notably Arrow's famous theorem [2]: It states that for at least three outcomes and two voters, no welfare function can simultaneously be Pareto-efficient (PE, all agents preferring o over o' must imply $o \succ_W o'$), independent of irrelevant alternatives (IIA, the relative ordering of o and o' does not change when agents change their preferences with respect to other outcomes), and non-dictatorial (ND, no single agent gets to determine the welfare ordering).

2.3 Related Work

There have already been efforts to combine soft constraints and voting. Most notably, the first algorithm to solve a problem specified with n c-semirings (precisely, fuzzy constraints) is *sequential voting* [7–9]: Agents vote sequentially over each variable's assigned domain value using social choice functions, according to a pre-defined ordering. The authors investigate voting-theoretical properties of their method. Specifically, the relationship between voting axioms assumed for *local* voting rules (i.e., voting over a single variable's assignment) to the global *solution* level was investigated. For instance, it is a necessary but not sufficient condition that all local rules be IIA in order for the global rule to be IIA as well. Also, if a single local rule is non-dictatorial, the global rule is non-dictatorial. Although their approach is mathematically appealing, in practice it suffers from the fine granularity that agents have to vote on: domain items for a single variable. This leads to myopic and overly optimistic estimations: Assuming, for example, that the combinations "(fish, white wine)" and "(meat, red wine)" are acceptable for an agent A with a slight preference for fish. Then A would place his or her bet on "white wine" although A might end up with the least desirable option "(meat, white wine)" since decisions cannot be retracted. Moreover, sequential voting may choose Pareto inefficient solutions deterministically even if the local rules are Pareto efficient (an example is provided in [23, Chap. 9]). Therefore, we propose to vote over solutions instead of individual variables' values.

Conversely, DCOP research is most prominently concerned with distributed settings and algorithms that operate across computational nodes, such as, e.g., the ADOPT algorithm [30]. Netzer and Meisels extended the classical sum-of-costs DCOP model (that, again, is equivalent to WCSP in soft constraints) to "distributed social constraint optimization problems" where social welfare functions replace summation [21]. Still, their approach calculates a single score for each assignment based on the agents' individual (numeric) valuations instead of a preference relation and can thus suffer from the bias problems shown in our introduction. Moreover, they assume some form of commensurability of utilities in the sense that an operator such as "maximize the unhappiest agent's value" is meaningful – if agents operate on distinct (esp. non-numerical) ordering relations, this is not obvious. In this paper, we abstract from the underlying distribution of the computational nodes or the specific distributed optimization algorithms and focus on adequate, unbiased *models* of "how to aggregate multiple agents' preferences" – as the first step towards more distributed solutions. Our experimental evaluations are thus conducted in a centralized setting.

Outcomes resulting from strategic interactions among several self-interested agents is central to game theory. Morgenstern and von Neumann introduced the foundations of numeric utility functions for ordinal preference relations in [19]. However, it is hard to consider bias resulting from such utility functions when multiple independent agents are involved. Mechanism design adds the strategic component of truth-telling to social choice situations [28]. We do not yet address such questions other than disincentivizing manipulation with weights.

The problem of bias reduction, in particular, has not yet been addressed with voting methods. Moreover, the existing proposals come with specialized implementations and are not readily available to end-users. Since our approach employs state-of-the-art constraint modeling languages, we expect it to inherit their benefits in terms of efficient solvers, user-friendliness, and flexibility. It is the first proposal in terms of applicable software and systems to offer access to social choice functions and discrete optimization.

3 Implementation

Our goal is to combine soft constraints and voting on the solution level. Formally, assume a CSP $= (X, D, C)$ and a set of agents $N = \{1, \ldots, n\}$ along with sets of soft constraints $(S_j)_{j \in N}$ mapping to PVS $(M_j)_{j \in N}$. Then the set of outcomes O corresponds to sols($[X \to D]$) and the preference profile $[\preceq] = (\preceq_j)_{j \in N}$ results from applying soft constraints: $\theta_1 \preceq_j \theta_2 \Leftrightarrow S_j(\theta_1) \leq_{M_j} S_j(\theta_2)$. Ideally, we can obtain a full social welfare ordering $W([\preceq])$, but we also settle for choice functions that return a single solution $C([\preceq]) \in [X \to D]$ as the group's favorite.

To implement this form of optimization for modeling languages, we first revisit how incremental search proceeds in a branch-and-bound fashion that systematically explores the full search space (we discuss extensions to local search later). For instance, in MiniSearch [22], we could write this as follows:

```
function ann: maximize_bab(var int: obj) =
   repeat(
       if next() then commit() /\
         post(obj > sol(obj))
       else break endif );
```

Any time the solver returns a solution, we can formulate new constraints to be propagated based on the current solution's values, e.g., to bound the objective. For instance, upon finding a solution with objective value `obj` $= 17$, we add a constraint `obj` > 17 to the constraint problem to find the next solution. If the resulting problem becomes unsatisfiable, we have found an optimal solution. More generally, that logic extends to arbitrary constraints for improvement which we refer to as `getBetter` predicates. MiniBrass [24] already generates these (hidden from the end-user) for atomic or complex objectives, including lexicographic and Pareto combinations. For instance, assuming two PVS M and N, solving for their Pareto combination leads to the following predicate:

```
predicate getBetterPareto(var M: overall_M, var N: overall_N) =
   post(  % both agents' PVS find the current solution worse or equal
     is_worse_or_equal_M(sol(overall_M), overall_M ) ) /\
     is_worse_or_equal_N(sol(overall_N), overall_N ) ) /\
     sol(overall_M) != overall_M \/ sol(overall_N) != overall_N) );
```

Our goal is to align voting methods with this optimization principle by generating `getBetter` constraints for them. Indeed, some voting methods are better suited for this task than others. For example, Borda voting would need to enumerate all, say, k solutions, rank the best solution with $k-1$, the next best with $k-2$ and so forth, for every agent. A priori, it is hard to guess the relative position any solution in the search space has as well as the size of $k = |\text{sols(CSP)}|$. By contrast, a variation of Condorcet's method is more "local" in the search space as it only requires pairwise comparisons, which we can exploit for optimization.

3.1 Condorcet Voting

Recall from Sect. 2.2 that canonical Condorcet voting proceeds by fixing an ordering of the outcomes O, comparing two adjacent options with respect to the pairwise majority, and keeping the winner. With the set of outcomes being the search space, ordering them in advance is, of course, infeasible. Nevertheless, we can tweak the method slightly for constraint-based optimization in MiniBrass:

1. Find the next solution (call it θ).
2. Impose a constraint that enforces that the *next* solution θ' must be preferred by a *majority* of voters and search for the next solution.
3. Repeat until no such solution can be found (or a cycle is detected).

In MiniZinc-style pseudocode, this predicate is generated as follows (M_i refers to the specific PVS element type of PVS i):

```
predicate getBetterCondorcet(array[1..N] of var Mi : overall) =
  % M_i represents the PVS of agent i
   post(
     % # agents that find the current solution worse than the next
     sum(i in 1..N) (bool2int(is_worse_i(sol(overall[i]), overall[i] ) )
     >
     % # agents that find the next solution worse than the current one
     sum(i in 1..N) (bool2int(is_worse_i(overall[i], sol(overall[i]) ) )    );
```

In fact, this is weakening the Pareto condition that *all* agents have to accept or prefer a new solution. If a solution is indeed a Condorcet-winner, this method will find it. If there is a Condorcet-cycle, there is no guarantee with respect to the outcome since the moment of termination depends on the ordering of the solutions as returned from the solver. However, such a cycle is easily detected by inspecting the trace of solutions. The only guarantee we can give upon termination is that there is no *unseen* solution that a majority of agents would prefer to the current one – which arguably still makes for a reasonable social choice function. The complexity of generating the above predicate is, analogously to Pareto and lexicographic combinations, hidden from MiniBrass end-users. For example, they would rewrite the `solve` expression in Fig. 2 as follows:

```
solve vote([alice, bob, carol], condorcet);
```

On a technical side-note, this approach requires the `getBetter` predicates to be *reifiable*, i.e., allow additional boolean variables to take their truth values.

3.2 Approval and Majority-Tops Voting

Arrow's theorem provides a hint for another suitable voting method to consider. Instead of allowing agents to order solutions arbitrarily, we can have them partition the search space into "acceptable" and "unacceptable" solutions, and search for a solution that is approved by the highest number of agents. Hence, we implement "approval voting" which turns out to have beneficial voting-theoretical properties: Approval voting is a non-dictatorial social welfare function that satisfies PE and IIA – due to the restriction to only two options, approved or disapproved;

Lemma 1. *With only approval or disapproval at hand, approval voting satisfies PE, IIA, and is non-dictatorial [28, p. 267].*

Arrow's theorem applies for votings with $|O| \geq 3$ and does therefore not apply in this restricted setting. Despite the restricted generality of having only two levels of satisfaction, a variety of use cases fall into that category (e.g., students proposing a personalized set of acceptable exam dates). Moreover, generating an optimization predicate (or even a numeric objective in this case) is straightforward. For a given assignment θ, we count the number of approving agents. Once we see a solution, we must impose a constraint that more agents approve of the next one. Since for approval voting, we know that the type of every agent's PVS type must be boolean, the `getBetter` predicate is simplified:

```
predicate getBetterApproval(array[1..N] of var bool: overallAgents) =
   post(
     sum(i in 1..N) (bool2int(overallAgents[i])) >
       sum(i in 1..N) (bool2int(sol(overallAgents[i])))   );
```

This results in a social welfare ordering over solutions. MiniBrass end-users would again only write

```
solve vote([alice, bob, carol], approval);
```

where MiniBrass would ensure that each submitted PVS for approval voting is indeed boolean-valued.

For more general orderings, approval voting is insufficient except if valuations are, e.g., thresholded. The most canonical threshold value imaginable is to approve a solution only if it evaluates to the top value ε_M in the corresponding PVS (e.g., 0 in WCSP, 1.0 for fuzzy CSP, or $\emptyset$ for a violation-set-based formalism). Solutions are then ranked according to the number of top-values. Since this is an adaption of the majority rule that asks for an outcome to count the number of agents that place it on top of their ranking, we call this variant *majority-tops*. We would then just count the number of agents that get all their wishes satisfied. In MiniBrass, we write, e.g.:

```
solve vote([alice, bob, carol], majorityTops);
```

To sum up, our proposed voting methods in MiniBrass encompass `condorcet`, `majorityTops`, `approval`, and (for numeric objectives) also `sumMin` and `sumMax`. The latter two sum up the overall valuations analogously to conventional WCSP or DCOP formulations.

3.3 Voting with Local Search

We want to conclude our proposed implementation with a word of caution. Larger problem instances can be prohibitive for systematic and complete search space traversal and require heuristic local search approaches such as large-neighborhood search. Arrow's theorem then proves practically relevant. More specifically, a social welfare function W violating IIA can lead to unexpected results: If n voters choose over a subset of the available solutions $\Theta \subseteq [X \to D]$, e.g., a neighborhood $\mathcal{N}(\theta) \subseteq [X \to D]$ of a solution θ, the local ordering obtained by applying W to the profile over Θ can be different from the ordering over Θ when the whole search space $[X \to D]$ is present. In practice, this means that even though the agents agree to switch from θ_1 to $\theta_2 \in \mathcal{N}(\theta_1)$ when voting over the neighborhood's options, they would rank θ_1 better than θ_2 if *all* solutions were up for election. Hence, an IIA welfare function such as approval voting is a more sensible choice for local search than Condorcet voting.

4 Experimental Evaluation

For our evaluation, we investigate two problems that are inspired by real-life decision situations faced at a typical university research group.

The first one, **lunch selection**, serves as our initial proof of concept and consists of deciding a shared meal plan during a research retreat. We assume a given set of prospective dishes and that a fixed-cardinality subset of those has to be selected for a week. Each voter has preferences concerning the presence or absence of certain dishes in the final selection.

The second one, **mentor matching**, is more complex than lunch selection and involves assigning students to industry mentoring partners where, for simplicity, we only consider students' preferences regarding companies. There are cardinality constraints on how many students each company can supervise.

In the real-life counterparts to the experimental setting proposed in this paper, agents were only asked to provide a partial order in MiniBrass and can leave out options. Here, we force them to submit a total order in both cases for better comparability. Some of the agents are allowed to cheat, i.e., to *amplify* their weights to gain an unfair advantage in DCOP/WCSP-style (summation-based) optimization. This *emulates* other less-obvious ways that introduce weight bias. Our goal is *to test if Condorcet voting can undo this artificial amplification.* Still, for the example problems, we could easily apply numerical normalization to undo the amplification effects. In more general settings involving non-numerical orderings, this would no longer be an option.

As a result of this forced total ordering over desirable outcomes, both instances give rise to an interpretable unit of satisfaction over all agents. This allows for measuring how well the proposed voting methods mitigate unfair preference specifications that we expect to influence the WCSP approach heavily. To quantify this in a controlled fashion, we also calculate a ground-truth baseline distribution of satisfaction values emerging from truthful, non-amplified weights from amplifying agents. We investigate several preference aggregation strategies (ignoring approval voting and majority-tops due to the ranked setting):

- **WCSP Unbiased:** Classical weight-based summed optimization with amplification deactivated
- **WCSP Biased:** Classical weight-based summed optimization with amplification activated
- **Condorcet:** asking for pairwise majority improvements
- **Pareto:** searching for Pareto-improvements

We hypothesize that the satisfaction distribution resulting from unbiased WCSP is similar to that of Condorcet voting whereas biased WCSP would strongly prefer the amplifier group. For both problems, we create 200 random instances (i.e., synthesized preferences). Since we observed qualitatively the same behavior for various parameter settings (e.g., the size of the restricted set of dishes or the number of students and companies) in both problems, we present the results for a fixed setting. For every instance, we pick a random subset of voters to become amplifiers, according to varying ratios. The allowed amplification factor is proportional to the number of agents, analogously to Fig. 1. All four aggregation methods are applied to the same 200 instances, to ensure comparability.

We solve lunch selection using Google OR-Tools 7.0 (CP solver) [16] and mentor matching using Chuffed [6], since we found these to be performing best for the respective problems. Each presented experiment runs on a machine having 4 Intel Xeon CPU 3.20 GHz cores and 14.7 GB RAM on 64 bit Ubuntu 16.04.[3]

4.1 Lunch Selection

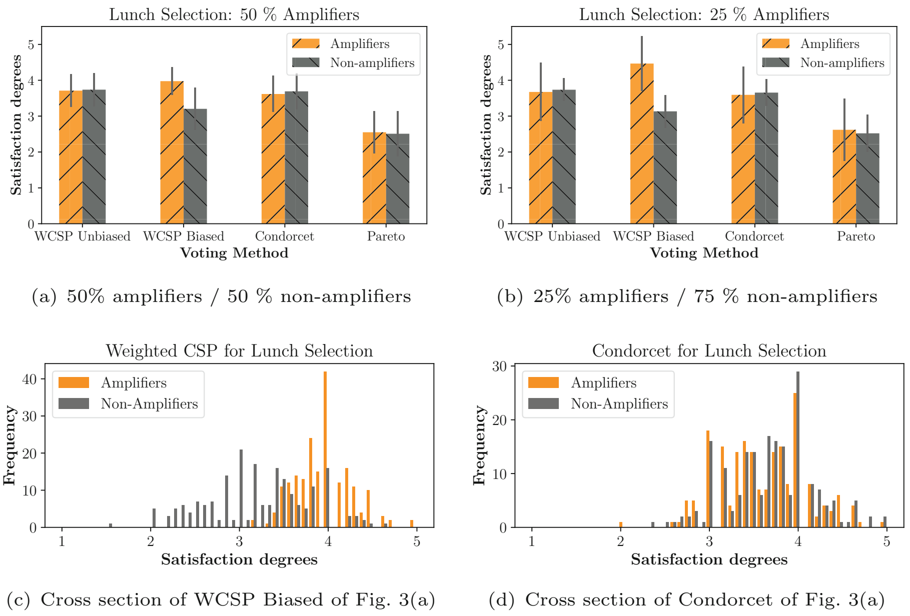
First, we consider the results obtained from the lunch selection experiment. Given the set of available dishes F, upon deciding the lunches in $L \subseteq F$, we can determine the satisfaction values per agents as follows: We assign one unit of satisfaction for every meal that agent i likes that is in L and, equivalently, one unit for every disliked meal which is not in L.[4] Agents specify their wishes as a ranking over F (not only approval sets), with the last positions corresponding to disliked dishes.

Figure 3 presents the average satisfaction degrees per group (amplifiers/non-amplifiers) obtained for this problem, once with 50% and once with 25% amplifiers. Figure 3(a) shows the results of equally splitting the agents into the groups of amplifiers and non-amplifiers. We can see that (as expected) the unbiased WCSP leads to an equal distribution of satisfaction degrees in both groups, whereas amplification clearly and unfairly treats amplifiers significantly better (plotted in Fig. 3(c), and also revealed by a student t-test at $\alpha = 10^{-3}$). However, that effect can almost entirely be reversed by using Condorcet voting instead that ends up having insignificantly varying satisfaction distributions (cf. Fig. 3(d)). This confirms the intuition presented in Fig. 1.

Besides, searching for Pareto improvements leads to significantly worse satisfaction degrees than Condorcet for both groups. This confirms our intuition that Pareto combinations alone are too indecisive, i.e., the society cannot make many improvement steps in the search tree since *all* agents need to agree on a better solution as opposed to Condorcet's method that only requires a *majority*

[3] The raw result data and experimental code can be found online at https://github.com/isse-augsburg/minibrass/tree/master/evaluation/minibrass-voting-experiment.

[4] In the experiments, $n = 12$ agents vote over a set L with $|L| = 4$ chosen from $|F| = 7$ available objects. This leaves us with only $\binom{7}{4} = 35$ solutions. Nevertheless, the bias reduction effects are already apparent in this small example.



(a) 50% amplifiers / 50 % non-amplifiers

(b) 25% amplifiers / 75 % non-amplifiers

(c) Cross section of WCSP Biased of Fig. 3(a)

(d) Cross section of Condorcet of Fig. 3(a)

Fig. 3. Lunch selection: Average satisfaction values per group over 200 instances. In the unbiased case, the amplification factor was deactivated.

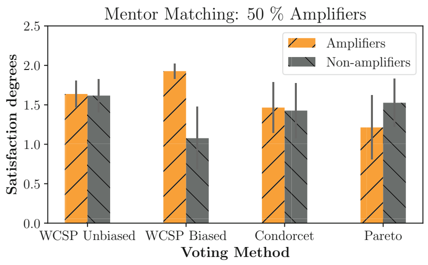
of them to do so. For a smaller set of amplifiers (25% instead of 50%), Fig. 3(b) presents qualitatively similar results. The amplifiers gain even more benefit in biased WCSP since there are fewer other amplifiers to split satisfaction degrees.

4.2 Mentor Matching

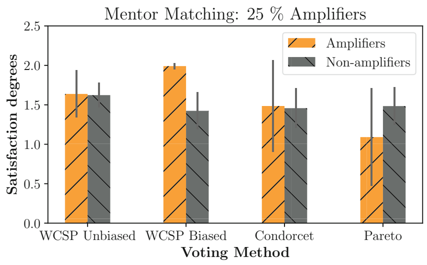
In mentor matching, we assign students to their mentoring companies. This gives us a natural measure of satisfaction, i.e., the rank of the assigned company, with $r_i = j$ denoting that agent i gets their j^{th} preference and $r_i = 1$ meaning top satisfaction. The minimal and maximal number of students a company can supervise are constrained which makes supervision a scarce resource.[5] The students' preferences are not arbitrary, but some companies had a higher probability of appearing higher than others – which corresponds to our real-life experiences.

Figure 4 shows overall results similar to the lunch selection case (we converted the students' achieved ranks to satisfaction degrees to have axes consistent with the previous example by subtracting ranks from a constant value). Condorcet voting mitigates the bias introduced by amplified weights, although the resulting average values are slightly lower than in the unbiased case. Interestingly,

[5] Here, $n = 18$ agents vote over assignments to six companies, with each company supervising at least two and at most three students. There are at most $6^{18} \approx 1.015^{14}$ solutions to explore, not accounting for the cardinality constraints.



(a) 50% amplifiers / 50 % non-amplifiers



(b) 25% amplifiers / 75 % non-amplifiers

Fig. 4. Mentor matching: Average satisfaction values per group over 200 instances. In the unbiased case, the amplification factor was deactivated.

Table 1. An illustrative example for skewed satisfaction resulting from Condorcet voting. Average value and sample standard deviation are denoted by $\bar{r}$ and σ_r.

Voting method	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	r_{16}	r_{17}	r_{18}	$\bar{r}$	σ_r
WCSP (Unbiased)	2	2	1	1	1	1	1	1	2	2	1	3	1	1	1	2	1	1	1.39	0.59
Condorcet	1	2	1	1	1	4	1	1	1	2	1	1	1	5	1	2	1	1	1.56	1.12

the average disadvantage of non-amplifiers is stronger in the 50% case depicted in Fig. 4(a) than in the 25% case shown in Fig. 4(b) whereas the average satisfaction of the amplifiers remains close to optimal. Pareto voting (unexpectedly) discriminates *against* amplifiers. This interesting result is due to the solver's default search strategy that favors smaller domains (i.e., the non-amplifiers' cost variables). Since Pareto search does not dive deeply into the search tree, we end up close to these biased solutions.[6]

We also note that, compared to lunch selection, the standard deviations of Condorcet voting are higher. Closer investigation of that issue reveals a weakness apparent in Condorcet voting that we exemplified with a case (ID 26 in the online results) in Table 1: While unbiased WCSP results in a rather fair allocation that never pairs an agent with a company worse than their third preference, Condorcet's method offers a solution that results in the fourth or even fifth preference for two students. Both assignments have similar average satisfaction degrees over all students, but we might consider Condorcet's result less fair. It is, however, a logical consequence of the focus on majority improvements: If all but two agents agree that (here) a mentoring assignment is better, it gets picked – even if this means substantial deterioration in satisfaction for the two agents. It does not make a difference *how strong* the dissatisfaction is.

To confirm this suspicion, in Table 2, we also show the distribution of sample standard deviations of satisfaction degrees per assignment as a rough measure of

[6] We confirmed that a different solver (Gecode [26]) is more balanced but still keep these unexpected results in the paper to highlight issues with Pareto optimization.

Table 2. Comparison of weighted CSP optimization (unbiased) and Condorcet voting. Sample standard deviations (S-STD) are metrics to measure each instance's "unfairness" and aggregated over all 200 instances. Analogously, SAT corresponds to satisfaction degrees as presented in Figs. 3 and 4.

Problem	WCSP SAT	WCSP S-STD	Condorcet SAT	Condorcet S-STD
Lunch Selection	3.72 (0.22)	1.19 (0.24)	3.64 (0.28)	1.19 (0.23)
Mentor Matching	1.62 (0.12)	**0.6** (0.14)	1.45 (0.18)	**1.02** (0.28)

unfairness (σ_r in Table 1). While this effect does not show up in lunch selection (due to the smaller search space), we can observe for mentor matching, regardless of how many amplifiers were present (therefore we only present the 50% amplifiers results). The average sample standard deviation of 0.6 for unbiased WCSP optimization is significantly lower than the value of 1.02 that Condorcet reaches. A student t-test at $\alpha = 10^{-3}$ confirms this for both amplifier ratios.[7]

5 Conclusion and Future Work

We presented an extension to conventional soft constraint optimization problems that allows to aggregate preferences using voting methods instead of either the usual numeric operations such as summation or Pareto/lexicographic combinations. This extension was able to "correct" the bias introduced by amplified weight specifications on two real-life-inspired problems to almost the level of unbiased specifications using Condorcet voting. Our evaluation revealed, however, that pure Condorcet voting is susceptible to producing unbalanced assignments at the expense of few agents due to its focus on the majority.

Using only ordinal information gives us no "metric" sense of different levels of dissatisfaction among a group of agents – we cannot relate discomfort a from agent A to discomfort b of another agent B. Therefore, a potential direction for research is to focus on methods that can produce more balanced assignments. It might still be necessary, for that matter, to introduce a numeric scale and allowing some form of transferable utility/budget or conversion rates. We hope to provide more variety in this type-constrained setting.

Additionally, we expect better fairness guarantees in repeated optimization settings (e.g., rostering problems over many weeks) as well as an "iterative deepening" approach that first votes over coarser classes of problems (using diverse solution search [17]) and subsequently refines those choices. Finally, we intend to implement our model formulation also in actual DCOP solvers that mostly need to be able to understand MiniZinc models. On a somewhat similar note, the scalability of the voting approach remains to be tested for larger models, perhaps

[7] Using the student t-test for significance is justified by observing that the sample standard deviations of satisfaction degrees follow a normal distribution according to a Shapiro-Wilk test at $\alpha = 10^{-3}$.

involving large neighborhood search. We expect many real-life problems close to end-users to benefit strongly from fairer voting methods than simple summation.

References

1. Andréka, H., Ryan, M., Schobbens, P.Y.: Operators and laws for combining preference relations. J. Log. Comput. **12**(1), 13–53 (2002)
2. Arrow, K.J.: Social Choice and Individual Values. Yale University Press, London (1951)
3. Arrow, K.J., Sen, A., Suzumura, K.: Handbook of Social Choice and Welfare, vol. 2. Elsevier, Amsterdam (2010)
4. Beldiceanu, N., Carlsson, M., Demassey, S., Petit, T.: Global constraint catalogue: past present and future. Constraints **12**(1), 21–62 (2007)
5. Bistarelli, S., Montanari, U., Rossi, F.: Semiring-based constraint satisfaction and optimization. J. ACM **44**(2), 201–236 (1997)
6. Chu, G.: Improving combinatorial optimization. Ph.d. thesis, University of Melbourne (2011)
7. Cornelio, C., Pini, M.S., Rossi, F., Venable, K.B.: Multi-agent soft constraint aggregation via sequential voting: theoretical and experimental results. Auton. Agent. Multi-Agent Syst. **33**(1–2), 159–191 (2019)
8. Dalla Pozza, G., Pini, M.S., Rossi, F., Venable, K.B.: Multi-agent soft constraint aggregation via sequential voting. In: Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011), pp. 172–177 (2011)
9. Dalla Pozza, G., Rossi, F., Venable, K.B.: Multi-agent soft constraint aggregation: a sequential approach. In: Proceedings of the 3rd International Conference on Agents and Artificial Intelligence (ICAART 2011), vol. 11 (2010)
10. Dekker, J.J., de la Banda, M.G., Schutt, A., Stuckey, P.J., Tack, G.: Solver-independent large neighbourhood search. In: Hooker, J. (ed.) CP 2018. LNCS, vol. 11008, pp. 81–98. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98334-9_6
11. Ehrgott, M.: Multicriteria Optimization, vol. 491. Springer Science & Business Media, New York (2005)
12. Fioretto, F., Pontelli, E., Yeoh, W.: Distributed constraint optimization problems and applications: a survey. J. Artif. Intell. Res. **61**, 623–698 (2018)
13. Fioretto, F., Yeoh, W., Pontelli, E., Ma, Y., Ranade, S.J.: A Distributed Constraint Optimization (DCOP) approach to the economic dispatch with demand response. In: Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017), pp. 999–1007. International Foundation for Autonomous Agents and Multiagent Systems (2017)
14. Frisch, A.M., Harvey, W., Jefferson, C., Martínez-Hernández, B., Miguel, I.: Essence: a constraint language for specifying combinatorial problems. Constraints **13**(3), 268–306 (2008)
15. Gadducci, F., Hölzl, M., Monreale, G.V., Wirsing, M.: Soft constraints for lexicographic orders. In: Castro, F., Gelbukh, A., González, M. (eds.) MICAI 2013. LNCS (LNAI), vol. 8265, pp. 68–79. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-45114-0_6
16. Google Optimization Tools (2017). https://developers.google.com/optimization. Accessed 29 June 2017

17. Hebrard, E., Hnich, B., O'Sullivan, B., Walsh, T.: Finding diverse and similar solutions in constraint programming. In: Proceedings of the 23rd National Conference Artificial Intelligence (AAAI 2005), vol. 5, pp. 372–377 (2005)
18. Meseguer, P., Rossi, F., Schiex, T.: Soft constraints. In: Rossi, F., van Beek, P., Walsh, T. (eds.) Handbook of Constraint Programming, chap. 9. Elsevier, Amsterdam (2006)
19. Morgenstern, O., von Neumann, J.: Theory of games and economic behavior (1944)
20. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: MiniZinc: towards a standard CP modelling language. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 529–543. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74970-7_38
21. Netzer, A., Meisels, A.: SOCIAL DCOP - social choice in distributed constraints optimization. In: Brazier, F.M.T., Nieuwenhuis, K., Pavlin, G., Warnier, M., Badica, C. (eds.) Intelligent Distributed Computing V. SCI, vol. 382, pp. 35–47. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24013-3_5
22. Rendl, A., Guns, T., Stuckey, P.J., Tack, G.: MiniSearch: a solver-independent meta-search language for MiniZinc. In: Pesant, G. (ed.) CP 2015. LNCS, vol. 9255, pp. 376–392. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23219-5_27
23. Schiendorfer, A.: Soft Constraints in MiniBrass: Foundations and Applications. Dissertation, Universität Augsburg (2019)
24. Schiendorfer, A., Knapp, A., Anders, G., Reif, W.: MiniBrass: soft constraints for MiniZinc. Constraints **23**, 403–450 (2018)
25. Schiex, T., Fargier, H., Verfaillie, G.: Valued constraint satisfaction problems: hard and easy problems. In: Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 1995), vol. 1, pp. 631–639. Morgan Kaufmann (1995)
26. Schulte, C., Lagerkvist, M.Z., Tack, G.: Gecode: generic constraint development environment. In: INFORMS Annual Meeting (2006)
27. Schulze, M.: A new monotonic, clone-independent, reversal symmetric, and condorcet-consistent single-winner election method. Soc. Choice Welfare **36**(2), 267–303 (2011)
28. Shoham, Y., Leyton-Brown, K.: Multiagent Systems: Algorithmic, Game-theoretic, and Logical Foundations. Cambridge University Press, Cambridge (2008)
29. Stuckey, P.J., Feydy, T., Schutt, A., Tack, G., Fischer, J.: The MiniZinc challenge 2008–2013. AI Mag. **35**(2), 55–60 (2014)
30. Yeoh, W., Felner, A., Koenig, S.: BnB-ADOPT: an asynchronous branch-and-bound DCOP algorithm. J. Artif. Intell. Res. **38**, 85–133 (2010)
31. Zivan, R., Yedidsion, H., Okamoto, S., Glinton, R., Sycara, K.: Distributed constraint optimization for teams of mobile sensing agents. Auton. Agent. Multi-Agent Syst. **29**(3), 495–536 (2015)