Choosing Combinatorial Social Choice by Heuristic Search

Minyi Li 1 and Quoc Bao Vo 2

Abstract. This paper studies the problem of computing aggregation rules in combinatorial domains, where the set of possible alternatives is a Cartesian product of (finite) domain values for each of a given set of variables, and these variables are usually not preferentially independent. We propose a very general heuristic framework SC* for computing different aggregation rules, including rules for cardinal preference structures and Condorcet-consistent rules. SC* highly reduces the search effort and avoid many pairwise comparisons, and thus it significantly reduces the running time. Moreover, SC* guarantees to choose the set of winners in aggregation rules for cardinal preferences. With Condorcet-consistent rules, SC* chooses the outcomes that are sufficiently close to the winners.

1 Introduction

In many multi-agent decision-making scenarios, the space of alternatives has a combinatorial structure: the set of possible alternatives is a Cartesian product of (finite) domain values for each of a given set of variables (aka. issues), and usually these variables are not preferentially independent. In classical social choice theory, candidates (aka. alternatives, outcomes) and the agents’ preferences are supposed to be listed explicitly as linear orders, and then a voting rule is applied to select one or a set of winning alternatives. These traditional methods rely on a demanding assumption that the candidates should not be too numerous. However, when the domain has a combinatorial structure, the number of alternatives is exponential in the number of variables, and therefore, the agents’ preferences are usually described in some compact representation languages rather than linear orders. This makes the social choice problem even more complex and challenging, because individual outcome comparisons (and thus pairwise comparisons between outcomes) in those languages might be computationally difficult. As most common aggregation methods need a number of operations at least linear (sometimes even quadratic or exponential) in the number of possible alternatives, generating the whole relation from those compact languages and directly applying rules to compute a social choice is impractical [10, 9].

Several ways of computing rules in combinatorial domains have been considered. The most straightforward way is to use issue-by-issue (a.k.a. seat-by-seat) sequential election. However, as soon as the variables are not preferentially independent, it is very likely that deciding on the issues separately will lead to suboptimal choices [4]. Many existing work consider imposing a domain restriction such as separability (which makes the issue-by-issue sequential election work), see e.g., [11]; or a weaker restriction such as O-legality [17], which allows for deciding on the issues one after another. Some later works [12, 16] relax those restrictions and introduce a notion of local Condorcet winners (a local Condorcet winner beats every other alternative that differs in a single variable from that local Condorcet winner). The authors also propose (and implement) algorithms for computing them. However, this notion of winner differs from a Condorcet winner in the alternative space, as it only takes into account neighbour alternatives.

In this paper, we introduce a very general heuristic framework SC* for computing social choice in combinatorial domains. SC* enables aggregating or voting on partial assignments (an assignment of a subset of variables; and possibly comparing assignments on different subset of variables) until all variables have been assigned a value. As a result, it neither requires a counting of candidates as that in most decision-making instances, nor imposes any restriction on the agents’ preference structures. The proposed heuristic approach allows searching in a much smaller sub-space of the alternatives, and thus requires significantly less pairwise outcome comparisons. It is general enough to be applicable to both aggregation rules for cardinal preferences and several Condorcet-consistent rules. Most importantly, SC* guarantees optimal social choice in aggregation rules for cardinal preference structures. With Condorcet-consistent rules, SC* chooses the candidates that are sufficiently close to the winners of the rule. Last but not least, the proposed algorithm is independent to the preference representations, and thus it is applicable to most representation languages in combinatorial domains. Notice that we omit all the theorem proofs in this paper due to space limitation, while a longer version containing all detail proofs can be found at http://www.ict.swin.edu.au/personal/myli/ecai2012.pdf

2 Preliminaries

Let $V = \{X_1, \ldots, X_n\}$ be a set of variables, where each variable $X_i$ takes values in a finite domain $D_{X_i}$. An alternative is uniquely identified by its values of all variables. The set of alternatives is denoted by $X$, such that $X = D_{X_1} \times \ldots \times D_{X_n}$. If $X = \{X_{i_1}, \ldots, X_{i_r}\} \subseteq V$, with $\sigma_1 < \cdots < \sigma_r$, then $D_\sigma$ denotes $D_{X_{i_1}} \times \cdots \times D_{X_{i_r}}$ and $\vec{x}$ denotes an assignment of variable values of $X$, i.e., $\vec{x} \in D_X$. If $X = V$, $\vec{x}$ is a complete assignment (corresponds to an outcome); otherwise $\vec{x}$ is called a partial assignment. If $\vec{x}$ and $\vec{y}$ are assignments to disjoint sets $X$ and $Y$, respectively ($X \cap Y = \emptyset$), we denote the combination of $\vec{x}$ and $\vec{y}$ by $\vec{x} \vec{y}$. If $X \cup Y = V$, we call $\vec{x} \vec{y}$ a completion of assignment $\vec{x}$. We denote by $\text{Comp}(\vec{x})$ the set of completions of $\vec{x}$.

A vote $p$ is a linear order on $X$, i.e., a transitive, antisymmetric, and total relation on $X$. We denote $\text{Li}(X)$ as the set of all possible linear orders on $X$. An $n$-agent profile $P$ is a collection of $n$ votes, that is, $P = \{p_1, \ldots, p_n\}$, where $p_i \in \text{Li}(X)$. Let $P(X)$ be the set of all possible profiles over $X$, a (voting or aggregation) rule $r : P(X) \rightarrow 2^X$ maps any profile $p \in P(X)$ to a subset of alternatives (winners).

Since direct assessment of the preference relations in combinato-

1 Swinburne University of Technology, Australia, email: myli@swin.edu.au
2 Swinburne University of Technology, Australia, email: BVO@swin.edu.au
3 The proposed heuristic framework

3.1 The search tree

We identify optimal social choice using a search tree \( T \). A search tree \( T \) can be considered as an assignment tree. For a combinatorial social choice problem over \( m \) variables \( V = \{X_1, \ldots, X_m\} \), let \( k \) be the maximum size of the variable domain: \( \forall x \in V, |D_x| \leq k \), then \( T \) is a \( k \)-ary tree. The depth of \( T \) is \( m \) with the root being at depth 0.

Suppose that a search tree \( T \) is generated following an order over variables \( \sigma = X_{\sigma_1}, \ldots, X_{\sigma_m} \); then each level \( \ell \) considers the value assigns to variable \( X_{\sigma_\ell} \). If a node \( \Phi \) from the upper level is being expanded with the value of a variable \( X \) at the current level, then \( \Phi \) has \( |D_x| \) branches and each branch assigns a different value \( x \) to \( X (x \in D_x) \). The root node represents an empty assignment. A node \( \Phi \) at depth \( \ell \) represents a unique value assignment (specified by the path from the root to that node) \( assg \in D_{X_{\sigma_1}} \times \cdots \times D_{X_{\sigma_m}} \), to the set of variables \( \{X_{\sigma_1}, \ldots, X_{\sigma_m}\} \). Each node at depth \( m \) corresponds to an alternative, as all the variables has been assigned a value.

3.2 The SC* search algorithm

Similar to other heuristic search algorithms [14], SC* creates a search tree by iteratively selecting a node that appears to be most likely to lead towards an optimal social choice (a winner of the rule). We first give a general definition of the evaluation function of the search strategy, and then we will specify different evaluation functions for different aggregation rules in the next section.

Definition 1 (Evaluation function) In an iteration, given a set of leaf nodes \( L \) in \( T \), an evaluation function \( F \) maps each node \( \Phi \) in \( L \) into a number \( F(\Phi : L \rightarrow \mathbb{R}) \), which indicates how promising (good) the node is, i.e., how likely the node will eventually lead to a winner of the given rule. In this work, \( F(\Phi) \) is modelled as kinds of dis-value of \( \Phi \) (e.g., cost, distance to goal, penalty, distrust). Therefore, the smaller the \( F \) value of a node \( \Phi \), the more promising \( \Phi \) is.

SC* (see Algorithm 1) is adapted from the A* heuristic search algorithm [6] with \( F \) being the heuristic function. We first randomly generate an order over variables \( \sigma = X_{\sigma_1} > \cdots > X_{\sigma_m} \); following which the search tree would be created (line 1). Starting with the root node with an empty assignment (logically expressed as \( \text{True} \)) (line 2), SC* maintains a priority queue of nodes to be expanded, known as the fringe. The lower \( F \) value of a node \( \Phi \), the higher its priority is, i.e., the more upfront it is in the fringe. We emphasize here that for two nodes with the same \( F \) value, the one at level \( \ell < m \) must be ordered before (given higher priority than) the one at level \( \ell = m \).

In each iteration of SC*, the first node \( \Phi \), i.e. the node with the lowest \( F \) value, is removed from the fringe (line 4). If it does not represent a complete assignment (\( assg \) denotes the assignment of \( \Phi \)), it will be expanded (line 6-9). Let \( X \) be the next variable to be assigned according to the order \( \sigma \), for each \( x \in D_x \), a child node expanded from \( \Phi \) with the assignment \( assg \wedge x \) will be created and added into the fringe. After creating all possible child nodes, the \( F \) value of each existing leaf node is computed accordingly, and the fringe is sorted in the ascending order of the \( F \) values; for two nodes with the same \( F \) value, the one with partial assignment (at level \( \ell < m \)) must be ordered before the one with a complete assignment (at level \( \ell = m \)).

Algorithm 1: SC*(\( P \))

| Input: | \( P \), a preference profile of \( n \) agents over \( m \) issues |
| Output: | \( outs \), a set of outcomes |

1. Randomly generate an order over variables \( \sigma = X_{\sigma_1} > \cdots > X_{\sigma_m} \);
2. fringe ← INSERT(MAKE-NODE(True), fringe);
3. while fringe ≠ ∅ do
   4. \( \Phi \) ← REMOVE-FIRST(fringe); assg ← ASSIGNMENT(\( \Phi \));
   5. if assg is not a complete assignment then
      6. \( X \) ← NEXT-VARIABLE(assg, \( \sigma \));
      7. foreach \( x \in D_x \) do
         8. INSERT(MAKE-NODE(assg ∧ \( x \)), fringe);
      end
   end
   9. Compute \( F \) for each leaf node;
   10. SORT-FRINGE-ASC(fringe);
   11. end
12. AppendTo(outs, assg);
13. \( \Phi' \) ← REMOVE-FIRST(fringe);
14. while \( F(\Phi') = F(\Phi) \) do
   15. assg ← ASSIGNMENT(\( \Phi' \));
   16. AppendTo(outs, assg);
   17. \( \Phi' \) ← REMOVE-FIRST(fringe);
18. end
19. end
20. return outs;
21. Notice that function SORT-FRINGE-ASC sorts the fringe according to an ascending order of the \( F \) values; for two nodes with the same \( F \) value, the one with partial assignment (at level \( \ell < m \)) must be ordered before the one with a complete assignment (at level \( \ell = m \)).

3.3 The basis of heuristic

All the heuristic search strategies introduced in this paper are based on the following definition of best possible alternative of a node:

Definition 2 (Best possible alternative) At each node \( \Phi \) of the search tree \( T \), each agent \( i \) has a best possible alternative (BPA) on \( \Phi \), denoted by BPA(\( \Phi \)), which is the optimistic outcome that agent \( i \) can obtain with the variable values assigned from the root to \( \Phi \) being fixed, i.e., the best outcome that agent \( i \) can obtain from the subtree of \( \Phi \). Formally,

\[ \text{BPA}(\Phi) = \text{OPTIMIZE}(assg, L) \]
where \( \text{assg} = \text{ASSIGNMENT}(\Phi) \) represents the assignment specified by the path from the root to \( \Phi \); function \( \text{OPTIMIZE} \) optimizes the values of the remaining variables that are not in \( \text{assg} \), according to agent \( i \)'s preference \( \Phi_i \).

Let \( \text{Comp}(\text{assg}) \) represents the completions of \( \text{assg} \), then \( \text{BPA}_i(\Phi) \) is the best alternative among \( \text{Comp}(\text{assg}) \) for agent \( i \). Consequently, the \( \text{BPA} \) of the root node for an agent \( i \) corresponds to the optimal (best) outcome for agent \( i \) in the entire outcome space, i.e. all variables are assigned the preferred values according to \( \Phi_i \). For a node \( \Phi \) at level \( m \) of \( T \), \( \forall i, j \in \{1, \ldots, n\} \) and \( i \neq j \), \( \text{BPA}_i(\Phi) = \text{BPA}_j(\Phi) = \text{ASSIGNMENT}(\Phi) \), as the assignment of \( \Phi \) is complete.

For instance, consider an ordering over three binary variables \( A \), \( B \) and \( C \): \( \text{abc} > \text{ab}c > \text{a}bc > \text{a}bc > \text{abc} > \text{abc} \), and a node \( \Phi \) with the assignment \( \text{assg} = a \) (resp. \( \text{assg} = \bar{a}b \)), the completions of \( \text{assg} \) is \{\( \text{abc}, \text{ab}c, \text{abc}, \text{a}bc \)\} (resp. \{\( \text{a}bc, \text{abc} \)). According to the preference order, the \( \text{BPA}_i(\Phi) \) is \( \text{abc} \) (resp. \( \text{abc} \)), because \( \text{abc} > \text{ab}c > \text{a}bc > \text{abc} \) (resp. \( \text{abc} \)).

### 4 Evaluation function of social choice rules

#### 4.1 Aggregation rules for cardinal preferences

When preferences are cardinal, the preference profile \( P = \langle f_1, \ldots, f_n \rangle \) consists of every agent’s scoring function. A scoring function \( f_i \) of an agent \( i \) maps every alternative \( x \in \mathcal{X} \) into a real number \( \mathbb{R} (f_i : \mathcal{X} \to \mathbb{R}) \), based on the penalty (or distance) \( x \) caused (or from the goal) according to agent \( i \)'s preference.

Typical examples of cardinal preference structures include penalty scoring functions, dis-utility functions and some logical preference languages like weighted goals and distance goals.

To model the preference of the group, for each alternative \( x \), the score \( f_i(x) \) of each agent \( i \) is synthesized by an aggregation operator \( \diamond : \mathbb{R}^n \to \mathbb{R} \) into a so-called social welfare function \( s(x) \) reflecting the preference of the group of agents [13]. Formally, given a cardinal preference profile \( P = \langle f_1, \ldots, f_n \rangle \), the social welfare scoring function \( s \) mapping from \( \mathcal{X} \) to \( \mathbb{R} \) is defined by:

\[
\forall x \in \mathcal{X}, s(x) = \diamond \langle f_i(x) \rangle \ {i = 1, \ldots, n}
\]

Classically, \( \diamond \) is an operator that satisfies non-decreasingness for each of its argument and commutativity. As discussed in [8], the most natural choices for \( \diamond \) are \( \text{sum} \) and \( \text{max} \). \( \text{sum} \) is a utilitarian aggregation operator, stating that the collective score of an outcome is the sum of the scores of the agents in the group. On the other hand, \( \text{max} \) states that the maximum score among all the agents should be considered. Thus, the \( \text{max} \) aggregation operator corresponds to the egalitarian social welfare. Finally, an outcome \( x \) is \( \diamond \)-optimal iff \( s(x) \) is minimized.

For each node \( \Phi \) in the search tree \( T \), each agent \( i \) has a best possible alternative \( \text{BPA}_i(\Phi) \), and accordingly, an optimistic score \( f_i(\text{BPA}_i(\Phi)) \), which is the best (the smallest) possible score that agent \( i \) may obtain from the subtree of \( \Phi \). Applying \( \text{SC}^* \) to compute an optimal social choice with cardinal preferences, the heuristic evaluation function \( \mathcal{F}_{\text{Cardinal}} \) is defined as follows.

#### Definition 3 (Evaluation function of cardinal rules)

The evaluation function \( \mathcal{F}_{\text{Cardinal}} \) mapping from a node \( \Phi \) to \( \mathbb{R} \), is defined by:

\[
\mathcal{F}_{\text{Cardinal}}(\Phi) = \diamond f_i(\text{BPA}_i(\Phi)) \ {i = 1, \ldots, n}
\]

In each iteration, \( \text{SC}^* \) chooses a node that has the minimum \( \mathcal{F}_{\text{Cardinal}} \) value to expand, until the node chosen for expansion corresponds to a complete assignment. Notice that with cardinal preference structures, the \( \mathcal{F} \) value of a node is fixed. Therefore, the \( \mathcal{F} \) value of a node will only need to be calculated once.

#### Theorem 1

Given that the social welfare function \( \diamond \) satisfies non-decreasingness and commutativity, the evaluation function \( \mathcal{F}_{\text{Cardinal}} \) is admissible.

Proof: On each node \( \Phi \) of the search tree, the evaluation function \( \mathcal{F}_{\text{Cardinal}} \) is aggregated from a set of optimistic (i.e. the possible minimum) scores of the agents, each of which results from the best possible alternative (BPA) of an agent. Travel along the path of the tree, both the individual and the collective scores are non-decreasing.

Consequently, for any node \( \Phi \) and \( \text{assg} = \text{ASSIGNMENT}(\Phi) \), it satisfies that \( \forall \mathcal{Y} \in \text{Comp}(\text{assg}) \) and \( \forall i \in \{1, \ldots, n\} \), \( f_i(\mathcal{Y}) \leq f_i(\bar{\mathcal{Y}}) \) where \( \bar{\mathcal{Y}} = \text{BPA}_i(\Phi) \). As \( \diamond \) satisfies non-decreasingness, \( \diamond f_i(\mathcal{Y}) \leq \diamond f_i(\bar{\mathcal{Y}}) \) if \( \mathcal{Y} = \bar{\mathcal{Y}} \), that is, \( \mathcal{F}_{\text{Cardinal}}(\Phi) \leq s(\bar{\mathcal{Y}}) \). Thus, \( \mathcal{F}_{\text{Cardinal}} \) never overestimates the collective scores of the leaf nodes and is admissible. □

#### Theorem 2

If \( \mathcal{F}_{\text{Cardinal}} \) is admissible, \( \text{SC}^* \) chooses the set of winners according to the rule \( \diamond \).

Proof: \( \text{SC}^* \) traverses the tree searching all neighbours; it follows the lowest evaluated value path and keeps a sorted priority queue of alternate path segments along the way. If at any point the path being followed has a higher evaluated value than other encountered path segments, the higher evaluated value path is kept in the fringe and the process is continued with the lower value sub-path. This continues until the current node chosen for expansion is a leaf node. Consequently, if the evaluation function \( \mathcal{F}_{\text{Cardinal}} \) never overestimates the collective scores of the leaf nodes, the first leaf node chosen for expansion that corresponds to an alternative is guaranteed to be optimal. □

**Example.** Consider three agents’ preferences over three binary domain variables \( V = \{A, B, C\} \) depicted in Figure 1. The scoring function \( f_i \) of an agent \( i \) is defined by the position of the alternative in the preference ordering, i.e., the optimal outcome being at 1 and the worst outcome being at 3. For instance, consider agent 1’s preference ordering in Figure 1(a), \( f_i(abc) = 1 \), \( f_i(abc) = 2 \), \( f_i(abc) = 8 \). The table below shows the BPA\( _i \) and \( f_i \) of each created node \( \Phi \) of each agent \( i \), and then the \( \mathcal{F}_{\text{Cardinal}} \) Values. In this example, we consider \( \text{max} \) rule (\( \diamond = \text{max} \)). Therefore, the \( \mathcal{F}_{\text{Cardinal}} \) value of a node \( \Phi \) (in a column) is the maximum \( f_i \) among the three rows.

<table>
<thead>
<tr>
<th>Agent</th>
<th>( \Phi_1 )</th>
<th>( \Phi_2 )</th>
<th>( \Phi_3 )</th>
<th>( \Phi_4 )</th>
<th>( \Phi_5 )</th>
<th>( \Phi_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>abc</td>
<td>abc</td>
<td>abc</td>
<td>abc</td>
<td>abc</td>
<td>abc</td>
</tr>
<tr>
<td>2</td>
<td>abc</td>
<td>abc</td>
<td>abc</td>
<td>abc</td>
<td>abc</td>
<td>abc</td>
</tr>
<tr>
<td>3</td>
<td>abc</td>
<td>abc</td>
<td>abc</td>
<td>abc</td>
<td>abc</td>
<td>abc</td>
</tr>
</tbody>
</table>

Figure 2 shows the search tree of this example. The 1st iteration creates two child nodes \( \Phi_1 \) and \( \Phi_6 \) of the root. \( \text{BPA}_1(\Phi_1) = abc \), \( f_1(abc) = 1 \); \( \text{BPA}_2(\Phi_1) = abc \), \( f_2(abc) = 5 \); \( \text{BPA}_1(\Phi_6) = abc \), \( f_5(abc) = 3 \). Therefore \( \mathcal{F}_{\text{Cardinal}}(\Phi_1) = \text{max}\{1, 5, 3\} = 5 \). Similarly,
4.2 Condorcet-consistent rules

Give a preference profile \( P \), a Condorcet winner (CW) is a candidate \( \tilde{x} \) such that when compared with any other candidate \( \tilde{y} (\tilde{x} \neq \tilde{y}) \), more than half of the agents prefer \( \tilde{x} \) over \( \tilde{y} \). Let \( \#i (i \leq n : \tilde{x} \succ i \tilde{y}) \) denote the number of agents who prefer an alternative \( \tilde{x} \) over another alternative \( \tilde{y} \), formally, an alternative \( \tilde{x} \) is a Condorcet winner iff \( \forall \tilde{y} \in X, \#i (i \leq n : \tilde{x} \succ i \tilde{y}) > \#j (j \leq n : \tilde{y} \succ i \tilde{x}) \).

When the Condorcet winner exists, it is unique. However, it is possible for a forum to exist in which, in collective preferences, can be cyclic (i.e., non-transitive), even if the preferences of individual agents are not. In combinatorial domains, however, the number of alternatives is often much larger than the number of agents, and therefore, there almost never exists a Condorcet winner in practical cases. Hence, in the next subsections, we consider two Condorcet-consistent rules, namely Copeland rule and Minimax rule. A rule is Condorcet-consistent, if it chooses the Condorcet winner when one exists. Before going to define the evaluation function of these two rules, we first introduce some related concepts as follows.

In each iteration, we denote \( L \) as the set of existing leaf nodes in the search tree. In order to apply \( S^5 \) to compute Condorcet-consistent rules, we first define an upper approximation of preference relations over the set of leaf nodes \( L \), based on the agent’s optimistic evaluation function (BPA) of each leaf node.

Definition 4 (Preference relations between leaf nodes) Given a pair of leaf nodes \( \Phi \) and \( \Phi' \), we say an agent \( i \) prefers \( \Phi \) over \( \Phi' \) (written as \( \Phi >_i \Phi' \)) iff an agent \( i \) prefers \( \Phi \) over the BPA of \( \Phi' \). Formally, \( \forall i \in \{1, \ldots, n\} \) and \( \forall \Phi, \Phi' \in L, \Phi >_i \Phi' \), if \( \Phi_{i, \Phi} = \Phi_{i, \Phi'} \) then \( \Phi_{i, \Phi} >_i \Phi_{i, \Phi'} \).

Proposition 1 Let \( \Phi \) be a node at level \( m \) (represents a complete assignment) and \( \Phi' \) be a node at level \( \ell \) \( (\ell \leq m) \), \( \tilde{x} \) be the alternative corresponds to \( \Phi \) (\( \tilde{x} = ASSIGNMENT(\Phi) \)) and \( ass' = ASSIGNMENT(\Phi') \), if an agent \( i \) prefers \( \Phi \) to \( \Phi' \), then \( \forall \tilde{y} \in Comp(ass') \), \( \Phi_{i, \Phi} >_i \Phi_{i, \Phi'} \).

Proof: \( \forall \tilde{y} \in Comp(ass') \). \( \Phi_{i, \Phi} >_i \Phi_{i, \Phi'} \) or \( \Phi_{i, \Phi'} >_i \Phi_{i, \Phi} \). As individual preference is transitive and \( \tilde{x} >_i \tilde{y}, \Phi_{i, \Phi} >_i \Phi_{i, \Phi'} \).

Definition 5 (Majority domination between leaf nodes) In an iteration, a leaf node \( \Phi \) dominates another leaf node \( \Phi' \), written as \( \Phi >_{maj} \Phi' \) if there is a majority number of agents who prefer \( \Phi \) over \( \Phi' \).

Corollary 1 Let \( \Phi \) be a node at level \( m \) and \( \Phi' \) be a node at level \( \ell \) \( (\ell \leq m) \): \( \tilde{x} = ASSIGNMENT(\Phi) \) and \( ass' = ASSIGNMENT(\Phi') \), if \( \Phi >_{maj} \Phi' \), then \( \forall \tilde{y} \in Comp(ass') \), \( \tilde{x} >_{maj} \tilde{y} \).

4.2.1 Copeland rule

Given a profile \( P \), the Copeland score of an alternative is the number of alternatives it beats in pairwise comparisons. Formally, for an alternative \( \tilde{x} \), let \( s(\tilde{x}) \) denotes the Copeland score of \( \tilde{x} \), then

\[ s(\tilde{x}) = \#(\tilde{y} \in X: \tilde{x} >_{maj} \tilde{y}) \]

A Copeland winner \( \tilde{x} \) is an alternative that maximizing \( s(\tilde{x}) \). In each iteration, we conduct a pairwise comparison over the set of leaf nodes \( L \) (based on Definition 4 and Definition 5). For consistency, we define the evaluation function as sort of disvalue of a node and the smaller \( F_{Copeland} \) is, the more promising the node is. Therefore, in an iteration, the \( F_{Copeland} \) value of a node \( \Phi \) is defined by the number of leaf nodes that majority dominates \( \Phi \).

Definition 6 (Copeland evaluation function) The evaluation function \( F_{Copeland} \) mapping from an existing leaf node \( \Phi (\Phi \in L) \) to \([0, +\infty]\), is defined by:

\[ F_{Copeland}(\Phi) = \#(\Phi' \in L: \Phi >_{maj} \Phi') \]

In the case with Copeland rule, the evaluation value \( F_{Copeland} \) of a node varies during the search (as some leaf nodes are expanded and new nodes are created). Therefore, in each iteration, we not only need to calculate the \( F_{Copeland} \) value of the new created nodes, but also need to update the \( F_{Copeland} \) value of other remaining leaf nodes.

Theorem 3 The evaluation function \( F_{Copeland} \) is not admissible.

Proof: For a node \( \Phi, F_{Copeland} \) is equal to the number of existing leaf nodes that dominates \( \Phi \). The domination relations between nodes is based on the agents’ BPA as nodes: \( \forall i \in \{1, \ldots, m\} \) and \( \forall \Phi, \Phi' \in L, \Phi >_i \Phi' \) if \( \Phi_{i, \Phi} >_i \Phi_{i, \Phi'} \). Moreover, \( \Phi >_{maj} \Phi' \) if there is a majority number of agents who prefer \( \Phi' \) over \( \Phi \). However, if \( \Phi >_{maj} \Phi' \), except the case that \( \Phi' \) is at level \( m \), it does not necessarily mean that there must exist an alternative in the subtree of \( \Phi' \) that can majority dominates \( \Phi \). For a simple example, consider three agents \( (1, 2, 3) \) and two binary variables \( X_1, X_2 \), \( D[X_1] = \{x_1, \bar{x}_1\} \), \( D[X_2] = \{x_2, \bar{x}_2\} \). The agents’ preferences are as follows:

- Agent 1: \( x_1 x_2 x_3 > \bar{x}_1 x_2 x_3, \bar{x}_1 \bar{x}_2 x_3 > x_1 \bar{x}_2 x_3 \)
- Agent 2: \( x_1 \bar{x}_2 x_3 > x_1 x_2 x_3, \bar{x}_1 \bar{x}_2 x_3 > \bar{x}_1 x_2 x_3 \)
- Agent 3: \( \bar{x}_1 x_2 x_3 > x_1 x_2 x_3, \bar{x}_1 \bar{x}_2 x_3 > \bar{x}_1 x_2 x_3 \)

Initially, there are only two nodes: \( \Phi \) with path assignment \( x_1 \) and \( \Phi \) with path assignment \( \bar{x}_1 \). Based on the agents’ preferences, then \( F_{Copeland}(\Phi') = 0 \) and \( F_{Copeland}(\Phi) = 1 \) as both agent 1 and 2 prefer \( \Phi' \) over \( \Phi \). However, \( BPA(\Phi') = x_1 x_2 \) and \( BPA(\Phi) = \bar{x}_1 x_2 \). Neither \( x_1 x_2 \) or \( \bar{x}_1 x_2 \) can majority dominates any completion of \( ASSIGNMENT(\Phi) \). The evaluation function \( F_{Copeland} \) may overestimate the number of alternatives that dominated a node, and it is not admissible.

Example (Cont.) We continue with the three agents’ preferences in Figure 1. In the table below we give a pairwise comparison matrix.
Given a profile $P$ and a pair of alternatives $x$ and $y$, let $N(x, y)$ denote the number of agents that rank $y$ ahead of $x$ in the profile $P$: $N(x, y) = \#\{i \leq n : y \succ_i x\}$. The Minimax score $s(x)$ of an alternative $x$ is defined by:

$$s(x) = \max_{\{i \leq n : y \succ_i x\}} N(x, y)$$

A Minimax winner $\overline{x}$ is an alternative that minimizing $s(\overline{x})$.

Similar to the Copeland rule, in each iteration, we conduct a pairwise comparison over the set of leaf nodes $L$. For a pair of leaf nodes $\Phi$ and $\Phi'$, we define $N(\Phi, \Phi')$ as the number of agents that prefer $\Phi'$ over $\Phi$: $N(\Phi, \Phi') = \#\{i \leq n : \Phi' \succ_i \Phi\}$. Then we can define the following evaluation function of Minimax rule:

$$f_{\text{Minimax}}(\Phi) = \max_{\{i \leq n : \Phi' \succ_i \Phi\}} N(\Phi, \Phi')$$

Theorem 4 The evaluation function $f_{\text{Minimax}}$ is not admissible.

Proof: For a node $\Phi$, $f_{\text{Minimax}}$ is the maximum number of defeats when compared with every other existing leaf node, and the number of defeats is counted based on the BPA of the nodes. The reasoning is similar to the case with Copeland rule. For any nodes $\Phi'$ and $\Phi$, even if a majority number of agents prefer a node $\Phi'$ to $\Phi$, it does not necessarily mean that this majority has the same BPA on $\Phi'$, i.e., it does not guarantee that there exists at least one alternative that is socially (majority) prefer to any alternative from the subtree of $\Phi$. Applied the same example in the proof for $f_{\text{Copeland}}$, we will also have $f_{\text{Minimax}}(\Phi') > s(\overline{x}, x_2) (\Phi$ is a node with path assignment $\overline{x}$ and $x_2 \in \text{Comp}(\Phi))$. Consequently, $f_{\text{Minimax}}$ may overestimate the number of worst case defeats of a node, and thus is not admissible. 

Example (Cont.) We apply $SC^*$ to compute the Minimax rule with our running example (Figure 1). In the table below we give the number of defeats in pairwise comparison ($\Phi$ in column and $\Phi'$ in row)

<table>
<thead>
<tr>
<th>$\Phi$</th>
<th>$\Phi'$</th>
<th>$\Phi_1$</th>
<th>$\Phi_2$</th>
<th>$\Phi_3$</th>
<th>$\Phi_4$</th>
<th>$\Phi_5$</th>
<th>$\Phi_6$</th>
<th>$\Phi_7$</th>
<th>$\Phi_8$</th>
<th>$\Phi_9$</th>
<th>$\Phi_{10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Phi_1$</td>
<td>$\times$</td>
<td>$1$</td>
<td>$1$</td>
<td>$1$</td>
<td>$-1$</td>
<td>$1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\Phi_2$</td>
<td>$1$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\Phi_3$</td>
<td>$-1$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\Phi_4$</td>
<td>$1$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\Phi_5$</td>
<td>$1$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\Phi_6$</td>
<td>$-1$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\Phi_7$</td>
<td>$1$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\Phi_8$</td>
<td>$1$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\Phi_9$</td>
<td>$1$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\Phi_{10}$</td>
<td>$1$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The evaluation function $f_{\text{Minimax}}$ is no longer a leaf node) in that iteration.
Figure 4 illustrates the search tree with this example (In the search tree \( F^3_{\text{Minimax}} \) is written shortly as \( F^3 \)). The search process is similar to the case with Copeland rule. However, in the 3rd iteration, \( F^3_{\text{Minimax}}(\Phi_1) = F^3_{\text{Minimax}}(\Phi_2) = F^3_{\text{Minimax}}(\Phi_3) = (2, \text{the minimum}) \). \( \Phi_1 \) is a partial assignment, it will be given higher priority than \( \Phi_2 \) and \( \Phi_3 \) in the \( \text{fringe} \). Therefore, \( \Phi_1 \) will be expanded in the 4th iteration. This process continues, until \( \Phi_3 \) is chosen for expansion (specifies a complete assignment). As, \( \Phi_3, \Phi_1, \Phi_2 \) and \( \Phi_0 \) have the same \( F^3_{\text{Minimax}} \) value, \( \text{SC}^* \) then returns four alternatives \( abc, ab\bar{c}, \bar{a}bc, \) and \( \bar{a}\bar{b}c \) (specified by \( \Phi_3, \Phi_2, \Phi_1, \Phi_0 \), respectively).

5 Experiment

In the experiments, we compare the proposed \( \text{SC}^* \) approach with two other algorithms: (i) a standard direct election method \( \text{DireE} \). \( \text{DireE} \) runs a direct election among all possible alternatives. It guarantees to find the winners of a rule. (ii) a sequential voting algorithm \( \text{SeqV} \). \( \text{SeqV} \) runs a sequential issue-by-issue voting following a random order over the set of variables. When the agents vote for the values of a remaining variable, they vote based on the best possible alternative with the variables that already assigned a value following being fixed.

In these experiments, we focus on binary variables and consider the representation language "SLO SCPnet" [5] to represent the agents’ preferences. The SLOSCPnets are generated with topology on the agents. The \( \text{SC}^* \) approach limits the number of visited nodes and the agents are not required to have common preference structures. For each number of variables, we run 5000 rounds of experiments.

Table 1. Experimental results with \text{max} rule for cardinal preferences

<table>
<thead>
<tr>
<th>Var.</th>
<th>Visited Nodes</th>
<th>Time (sec)</th>
<th>Succ. R. (%)</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>29.5</td>
<td>0.15</td>
<td>0.07</td>
<td>0.11</td>
</tr>
<tr>
<td>10</td>
<td>94.0</td>
<td>10.17</td>
<td>0.29</td>
<td>0.71</td>
</tr>
<tr>
<td>15</td>
<td>200.7</td>
<td>446.6</td>
<td>0.61</td>
<td>2.06</td>
</tr>
</tbody>
</table>

With aggregation rules for cardinal preferences, we consider \( \text{max} \). Table 1 shows the experimental results with 5 to 15 variables. It can be clearly seen that \( \text{SC}^* \) limits the number of visited nodes and the running time of \( \text{SC}^* \) is reduced by several orders of magnitude compared to the \( \text{DireE} \) algorithm when the number of variables is large. On the other hand, the running time for \( \text{SeqV} \) algorithm is the least among the three algorithms. However, the percentage of cases it chooses a winner is very low. The last two columns show the success rate (Succ. R.) and the distance between the winners and the outcome chosen by \( \text{SeqV} \) algorithm. The distance is defined by the difference between the social welfare scores of the winner and the outcome chosen by \( \text{SeqV} \) algorithm. When the number of variables is large, for instance 15 variables, in less than 4% of these experiments, the \( \text{SeqV} \) can find out a winner. Also, the average distance to the winners is quite large, 4096.0 in the case of 15 variables.

For the experiments with Condorcet-consistent rules, Table 2 and Table 3 shows the results with Copeland and Minimax rule, respectively. On the one hand, \( \text{SC}^* \) is much faster than the \( \text{DireE} \) algorithm, which becomes infeasible when the number of variables is larger than 8. On the other hand, \( \text{SC}^* \) provides a much higher success rate (Succ. R.) and smaller distance to the winners than the \( \text{SeqV} \) algorithm. Here, the distance to a Copeland winner (resp. a Minimax winner) is the difference of the Copeland scores (resp. Minimax scores) between the winners and the outcomes chosen by the algorithms.

Table 2. Experimental results with Copeland rule

<table>
<thead>
<tr>
<th>Var.</th>
<th>Avg. PC Reduction</th>
<th>Time (sec)</th>
<th>Succ. R. (%)</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>20.4%</td>
<td>0.03</td>
<td>0.01</td>
<td>0.07</td>
</tr>
<tr>
<td>6</td>
<td>67.0%</td>
<td>0.03</td>
<td>0.29</td>
<td>21.8%</td>
</tr>
<tr>
<td>8</td>
<td>89.4%</td>
<td>0.04</td>
<td>0.97</td>
<td>8.7%</td>
</tr>
</tbody>
</table>

PC Reduction: reduction in the number of pairwise comparisons.

Table 3. Experimental results with Minimax rule

<table>
<thead>
<tr>
<th>Var.</th>
<th>Avg. PC Reduction</th>
<th>Time (sec)</th>
<th>Succ. R. (%)</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>14.9%</td>
<td>0.87</td>
<td>0.01</td>
<td>0.06</td>
</tr>
<tr>
<td>6</td>
<td>50.3%</td>
<td>22.92</td>
<td>0.02</td>
<td>0.28</td>
</tr>
<tr>
<td>8</td>
<td>83.6%</td>
<td>517.6</td>
<td>0.04</td>
<td>1.0</td>
</tr>
</tbody>
</table>

PC Reduction: reduction in the number of pairwise comparisons.

6 Conclusion and future work

We have studied the problem of combinatorial social choice in this paper. As the size of alternative space is huge in combinatorial domains, we proposed a very general heuristic framework \( \text{SC}^* \) for computing different aggregation rules. \( \text{SC}^* \) guarantees to choose the winners of the rules for aggregating cardinal preferences. In the cases with Condorcet-consistent rules, \( \text{SC}^* \) chooses the alternatives that are sufficiently close to the winners.

When aggregating cardinal preferences, by considering each agent’s score value as an objective, \( \text{SC}^* \) processes in a way similar to the heuristic algorithm \( U^* \) [15] for multi-objective optimal path searching in acyclic OR-graphs. However, there are significant differences between the two:

i) In the problem of optimal path searching in OR-graph, \( U^* \) is a best-first search algorithm in which the heuristic information of a node is assumed given. \( U^* \) algorithm aggregates the given objective values (called reward vectors) of the path from the start node to the current node and the estimated reward vectors from the current node to the end node (the assumed given heuristic information) into a utility value \( u \) to guide the search. On the other hand, our proposed \( \text{SC}^* \) algorithm is designed to aggregate multi-agent preferences in combinatorial domains. \( \text{SC}^* \) algorithm defines the admissible heuristic based on the best possible alternative (BPA) of each agent. Consequently, we have proposed a method to obtain an admissible heuristic value based on the agents’ preferences instead of assuming that a heuristic function is given.

ii) The decisions for node expansion in the algorithms \( U^* \) and \( SC^* \) are different. \( U^* \) algorithm selects among the nodes based on a function \( IE \), which calculates an upper bound utility among all possible options of a node (e.g., possible next arcs to take or possible paths from the node to the end). On the other hand, instead of having to consider all possible options, the evaluation function used by \( SC^* \) (see Definition 3) aggregates the agents’ scores for their optimistic paths from the current node (i.e., the scores of their BPA) keeping in mind the scores of the BPA’s. Note that for a given node, calculating the BPA (i.e., optimal path) for an individual agent is computationally easy for acyclic preference structures. Hence, \( SC^* \) essentially does not consider all possible paths from the current node to the end node. In stead, it computes the evaluation value based on the individual agents’ scores.

iii) With preferences in combinatorial domains, variables are interdependent, and an individual agent’s preference (and thus the collective preference) over the value of a variable may depend on the values assigned to some other variables. As a result, different from the problem discussed in [15], the cost (or reward vector) for a node is not a fixed vector. In a combinatorial social choice problem, the evaluation
value of a node depends on its ancestors on the search tree as well as the variables to be assigned before a goal node is reached.

iv) Finally, the operator \(\ast\) used in [15] for aggregating reward vectors of multiple arcs or sub-paths is assumed to be order-preserving. On the other hand, due to the interdependency between variables in the context of preference, it is not the case when aggregating preferences in combinatorial domains.

SC* is based on the agents' optimistic evaluations of alternatives, i.e., best possible alternatives of the nodes. As an issue of future research, it would be interesting to further investigate the pessimistic evaluations, i.e., worst possible alternatives of nodes. A further extension of this work may take into account global constraints (which makes some alternatives not available). Last but not least, other ways to model admissible evaluation functions for Condorcet-consistent rules are also an important topic of our future research.

7 Acknowledgement

We would like to thank the anonymous reviewers for fruitful discussions and comments. This work was partially supported by the ARC Discovery Grants DP0987380 and DP110103671.

REFERENCES


