Comparing direct algorithms for two-player fair division of indivisible items – A computational study

D. Marc Kilgour
Wilfrid Laurier University
Waterloo, Canada
mkilgour@wlu.ca

Rudolf Vetschera
University of Vienna
Vienna, Austria
rudolf.vetschera@univie.ac.at

July 5, 2017

Abstract

We present an exhaustive computational study of algorithms for two-person allocation of indivisible objects. We identify eight algorithms that generate balanced allocations using only players' ordinal rankings and test them over all possible rankings of up to $N = 12$ items. Algorithms are evaluated according to the fairness and efficiency properties of the allocations they produce, including the ordinal properties of envy-freeness, Pareto-optimality, maximinality, and three similar properties based on Borda count. Our results indicate that, overall, the algorithms manage to identify a large majority of allocations that exhibit all three of the ordinal properties. Algorithms that combine bottom-up and top-down approaches find at least one good allocation whenever one exists, while other algorithms fail sometimes. Performance is not as good for Borda properties: In about 6.6% of the problems in which such an allocation exists, no algorithm can find it. We provide examples that illustrate how algorithms behave, and how they fail. We also discuss the overlap between allocations found by different algorithms, and find that algorithms combining both directions are able to find most of the allocations found by unidirectional algorithms, although there are many instances of allocations that can be found only by an algorithm in the latter group.
Keywords: Fair division; Indivisible items; Borda score; Computational study
JEL classification: C63, D63

1 Introduction

When a resource must be shared among individuals, questions of fairness and efficiency inevitably arise. The allocation must take into account which portions of the resource might be assigned to each individual, as well as how individuals

value the various portions they may receive. But a more fundamental issue is what kind of distribution is "fair," as well as whether and how fairness can be achieved in a given context. Fair solutions may be easy to find, or difficult, or might not exist at all. The well-known *I cut, you choose* procedure for two-person cake division is an example of a straightforward algorithm that guarantees fairness in a particular context—an infinitely divisible resource, to be allocated to two individuals with no knowledge of each other's preferences. For general overviews of fair division problems and applications, see Thomson (2016); Klamler (2010); Young (1994).

When the resource consists of a finite number of objects that cannot be divided—and therefore cannot be shared—fair division problems often become more difficult. Each agent must receive a subset of the objects, often called a *bundle*. One difficulty is that, even though the number of possible allocations is finite, combinatorial effects imply that it can be very large, making computability an issue. A second major difficulty is the inherent granularity of the problem; it may be that no allocations exist with desired fairness properties such as envy-freeness (no individual prefers another's bundle to his or her own). For a broad review of the fair division of indivisible objects, see Bouveret et al. (2016).

Any reasonable method for allocating a set of indivisible items must take some information about individuals' preferences into account. Given the difficulties of ranking a very large number of possible bundles, approaches to fair division of indivisible items often rely on preferences over individual items, which are then "lifted" (Bouveret et al., 2016) to the level of bundles. In the simplest case, utilities are linear, meaning that there are no synergies, positive or negative, between or among objects. Even in this case, the Santa Claus problem—to distribute a finite number of gifts to a finite number of children so as to maximize the utility of the unhappiest child—is NP-hard, and cannot even be approximated efficiently (Bansal and Sviridenko, 2006).

Despite the inherent difficulty of the problem, many algorithms that find allocations with at least some fairness properties have been developed in the past few years. The aim of this paper is to compare these algorithms. In line with much of the existing literature, we focus on algorithms that are specifically designed for allocation to two individuals whose preferences are linear, and that produce balanced allocations (each individual receives the same number of objects). With this restriction, it is possible to determine whether an allocation is envy-free using only each individual's preference ordering over objects; moreover, an allocation that is envy-free in this ordinal sense will be envy-free for any cardinal utilities that respect the ordering. Below we will discuss other criteria for fairness that apply to the two-person, balanced, ordinal case.

Although the restriction to only two players may seem severe, this setting captures some practical problems such as division of marital property in a divorce or division of an estate between two heirs. Indeed, many algorithms have been proposed to find a balanced allocation of indivisible objects to two individuals. This is also an important special case, in the sense that a good algorithm for two-person balanced allocation may provide important clues about how to

solve more general problems of allocating indivisible items, such as workers to tasks or aircraft to landing slots.

The main purpose of this paper is to compare these algorithms. We focus exclusively on algorithms that allocate all items in one pass over the players' rankings, i.e., they consider each item only once and immediately decide to which player it should be allocated. In particular, we do not include algorithms that leave a "Contested Pile" of unallocated items to be distributed (perhaps by a different algorithm) in a second stage. Below we compare eight algorithms, five from the literature plus three extensions developed in the course of this project. We compare them in a computational study using various criteria of fairness and efficiency. We apply criteria based on ordinal preferences, and complement them by additional criteria based on Borda scores. Borda scores, which can be considered a simple approximation to cardinal utilities (Darmann and Klamler, 2016), allow for additional insights into the properties of the algorithms.

Our study is exhaustive in several ways. First, for each problem size we study, we consider all possible problems (i.e., all possible profiles of individual rankings). Because we consider every problem of a given size, we can make exact statements about how often a desirable allocation is found by each algorithm, rather than the probabilistic statements that would follow from a random sampling approach. Second, for each problem we consider not only the allocations found by each algorithm, but also all possible balanced allocations. This approach allows us to identify interesting allocations that are not found by any particular algorithm (and perhaps not by any algorithm), and generally allows us to measure how often algorithms fail. Finally, we consider all allocations that an algorithm might find for a given problem. All the algorithms we study require one or more arbitrary choices at some point during execution (e.g., in algorithms that consider players in turns, one player must be considered first). We believe that it is important to assess an algorithm's performance on a given problem not just by one (arbitrarily selected) allocation that it might produce, but by all such allocations. Furthermore, we see the frequency of branching during execution of an algorithm, and the number of allocations it generates on average and in extreme cases, as highly relevant to its application to practical problems.

This is a computational study, and the goal of providing an exhaustive analysis restricts our effort to relatively small problems. The requirement of balanced allocations to two individuals implies that the number of items to be allocated must be even; we consider allocations of 4, 6, 8, 10, and 12 items. Although our size restriction is quite severe, we nonetheless consider the analysis of such small problems a useful exercise. First, even within the range of problems we analyze, we are able to find clear differences in algorithms, and can discern the tendencies of some algorithms to perform markedly better, or worse, than others. Moreover, in small problems it is often easy to understand which underlying characteristics of an algorithm lead to which effects. Furthermore, the effects of indivisibility are obviously stronger when the number of items is small, so we believe that the specific properties of algorithms are mostly likely to manifest themselves in small problems. Finally, many real-world problems of

allocation (such as allocation of cabinet seats to political parties) involve only a small number of items.

The paper proceeds with a discussion of fairness properties of allocations to be used in comparing algorithms (Section 2), followed by a description of the algorithms and their implementation, usually accompanied by pseudo-code (Section 3). Then (Section 4) we elaborate on exactly how we compare algorithms, and formulate the detailed research questions we want to study. After presenting our results in detail (Section 5), we draw some conclusions (Section 6), including some ideas about important questions that remain unanswered.

2 Properties of allocations

We consider a set S of $N = |S|$ items to be distributed to two players in the set $M = \{A, B\}$. For a player $m \in M$, denote the opponent of m by $\overline{m}$. An allocation X assigns subsets $X_A \subseteq S$ to A and $X_B \subseteq S$ to B, such that $X_A \cap X_B = \emptyset$ and $X_A \cup X_B = S$. We assume throughout that N is even and consider only balanced allocations, i.e., we require $|X_A| = |X_B| = N/2$. Denote the set of all balanced allocations of S by $\mathcal{A}$.

To evaluate allocations, one needs a model of players's preferences over sets of items. Here we focus on models of preferences on subsets of S that can be obtained from a (strict) ranking of S. Denote player m's ranking of items by $\succ_m$. We construct a partial ordering of subsets of S called *ordinally less* (Brams et al., 2012). A set of items X_m is *ordinally less* than a set Y_m for player m if there exists an injective mapping $f : Y_m \to X_m$ so that $\forall y \in Y_m : y \succ_m f(y)$. We denote the relation *ordinally less* for player m by $\prec^o_m$ (and the analogously defined *ordinally more* by $\succ^o_m$). The relation ordinally less is sufficient to define several fairness or efficiency properties for allocations. Of course, if a cardinal evaluation of items is available (even if it is not comparable between players), many more properties of allocations can be defined (Bouveret and Lemaître, 2016).

Pareto optimality is an efficiency property. An allocation X is *Pareto optimal (PO)* if there is no other allocation Y such that, for both players, the set of items received under X is ordinally less than the set of items received under Y, i.e. allocation X is *PO* iff

$$\nexists Y \in \mathcal{A} : \forall m \in M : Y_m \succeq^o_m X_m \wedge \exists n \in M : Y_n \succ^o_n X_n \tag{1}$$

Note that we define Pareto optimality in the usual way based on weak dominance. Nonetheless, when balanced allocations are compared, only strict dominance is possible because any change in a balanced allocation means that both players' sets of items must change. Thus, assuming that players have strict rankings of items, then no player can be indifferent between the sets of items received under two different allocations.

The second property we consider is *envy-freeness (EF)*. An allocation X is envy-free if no player finds the set of items received to be ordinally less than its

complement (which is, of course, allocated to the opponent (Klamler, 2010)). An allocation X is EF iff

$$\forall m \in M : X_m \succ_m^o X_{\overline{m}} \tag{2}$$

As another measure of fairness of allocations, we use the max-min property. An allocation is *max-min (MM)* if the maximum rank of any item allocated to a player in that player's ranking is minimal (Brams et al., 2016). Denote by $r_m(i)$ the rank of item i in the ranking of player m, taking the most preferred item to have rank 1 and the least preferred item to have rank N. An allocation X is MM iff

$$\max_{m \in M} \max_{i \in X_m} r_m(i) = \min_{Y \in \mathcal{A}} \max_{m \in M} \max_{i \in Y_m} r_m(i) \tag{3}$$

Although these properties are commonly used in the literature on fair division of indivisible items — in fact, many of the algorithms we study were specifically designed to fulfill some of them — they are not without drawbacks for the purposes of our study. Most notably, ordinally less is not a complete relation on the set of all (balanced) allocations. Several properties can be established only by verifying that no element satisfying a particular relation exists (e.g., to show that Pareto optimality fails, another allocation must be found that provides preferred subsets to both players; to show that envy-freeness fails, there must be a player who prefers the opponent's subset to the player's own). In such cases, the condition is not violated unless there is a *comparable* subset that is ordinally less or ordinally more, as required. One could argue that a complete relation would provide more potential violations, so an incomplete relation is a relatively weak basis for assessing allocations.

We therefore complement the properties defined using the *ordinally less* relation with properties based on an order relation on sets of items that holds for more pairs of subsets, namely Borda scores. When only ordinal information is available, Borda scores constitute a reasonable approximation to cardinal utilities (Darmann and Klamler, 2016, p.545).

We define four properties of allocations based on their Borda scores, analogous to the properties based on the ordinally less relation. Denote the Borda score of a set X_m allocated to player m using player m's ranking of items by $B_m(X_m)$, defined as follows:

$$B_m(X_m) = \sum_{i \in X_m} (N + 1 - r_m(i)) \tag{4}$$

A balanced allocation X is *Borda Pareto optimal (BP)* if no other balanced allocation Y exists such that, for both players, the Borda score of items received under Y is greater or equal the Borda score of the items received under X and, for at least one player, strictly greater. An allocation X is therefore BP iff

$$\nexists Y \in \mathcal{A} : \forall m \in M : B_m(Y_m) \geq B_m(X_m) \wedge \exists n \in M : B_n(Y_n) > B_n(X_n) \tag{5}$$

Using Borda scores, we can identify an even stronger measure of efficiency, one that directly relates to the utilitarian concept of efficiency as score maximization (Bertsimas et al., 2012). We define the property of *maximal Borda sum*

(BS) by

$$\sum_{m \in M} B_m(X_m) = \max_{Y \in \mathcal{A}} \sum_{m \in M} B_m(Y_m) \tag{6}$$

Because BS clearly implies BP, we concentrate on the BS property.

The concept of envy-freeness can be extended directly to the case of cardinal evaluations by requiring that no player assigns a higher (cardinal) utility to the set of items allocated to another player than to his or her own allocation (Bouveret and Lemaître, 2016). The same approach can also be applied to Borda scores. In the case of two players, this requirement is equivalent to the concept of proportionality introduced for Borda scores by Darmann and Klamler (2016). In general, proportionality requires that each player receives a utility of at least $V/|M|$, where V is the utility that the player assigns to the set of all items. In the case of $|M| = 2$ players, it is clear that this condition is fulfilled only if each player's utility for his or her subset is more than half of the total utility of all items, which means that the player must evaluate the set of items allocated to the opponent as less than one half of total utility (assuming utilities are additive, which precludes positive or negative synergies between or among items).

We therefore define an allocation X to be *Borda-proportional* (or equivalently in the case of two players *Borda-envy-free (BE)*) iff

$$\forall m \in M : B_m(X_m) \geq B_m(X_{\overline{m}}). \tag{7}$$

We denote this property BE in order to highlight its similarity to envy-freeness in the ordinal case.

Because the max-min property MM refers to ranks of items, it can be readily extended to Borda scores. We define an allocation X to be *Borda max-min (BM)* iff

$$\min_{m \in M} B_m(X_m) = \max_{Y \in \mathcal{A}} \min_{m \in M} B_m(Y_m). \tag{8}$$

Thus an allocation that satisfies BM maximizes the Borda score of the worse-off player.

3 Algorithms

3.1 Notation

The algorithms we consider take the players' rankings as input and process them so as to assign items in a balanced way. At any point in the execution of the algorithm, denote the set of items already allocated to player m by $Z_m \subset S$, and the set of as yet unallocated items by $U \subseteq S$. Obviously, $Z_A \cap Z_B = \emptyset$, $Z_m \cap U = \emptyset$ for all $m \in M$, and $Z_A \cup Z_B \cup U = S$.

For any set of items $V \subseteq S$ and any player $m \in M$, we define $top_m(V)$ to be player m's most preferred item in V, i.e.

$$top_m(V) = \arg\min_{i \in V} r_m(i), \tag{9}$$

and $last_m(V)$ to be player m's least preferred item in V,

$$last_m(V) = \arg\max_{i \in V} r_m(i). \tag{10}$$

Similarly, if $|V| > 1$, we denote player m's second most preferred item in V by $sb_m(V)$, where

$$sb_m(V) = \arg\min_{i \in V \setminus \{top_m(V)\}} r_m(i). \tag{11}$$

Also, the set of unallocated items that player m ranks at rank l or better is

$$H_m(l) = \{i \in U : r_m(i) \leq l\} \tag{12}$$

In the descriptions of algorithms that follow, we will write "*allocate* (i, j)" to indicate that item i is allocated to player A (i.e., $Z_A := Z_A \cup \{i\}$) and item j to player B ($Z_B := Z_B \cup \{j\}$), and that U is updated accordingly ($U := U \setminus \{i, j\}$).

Algorithms that consider the players simultaneously might require arbitrary decisions during the execution of the algorithm. Thus, for any particular input, an algorithm may take several paths and potentially deliver more than one allocation. Because one goal of this study is to analyze *all* allocations that an algorithm might generate, algorithms are implemented as recursive procedures, taking as arguments the current allocations to players as well as the current set of unallocated items. For short, we will write "*allocate* (i, j) *and continue*" to indicate that items i and j are to be allocated as indicated above, and then a new recursive call to the algorithm is to be made using the updated values of Z_A, Z_B, and U. Note that if multiple allocations are possible at any point in the execution of the algorithm, the next level of recursion is invoked more than once with different values of Z_A, Z_B, and U.

3.2 Original Sequential Algorithm (OS)

This algorithm was introduced by Brams et al. (2015) under the name Sequential Algorithm (SA). Since we also consider a modified version of that algorithm in our study, we here refer to it as the "Original SA" algorithm, or OS for short. It generates allocations that are Pareto-optimal and envy-free, provided an envy-free allocation exists. But allocations generated by OS are not guaranteed to satisfy the max-min property.

In addition to the partial allocations and the set of unallocated items, the recursive procedure for this algorithm has a third parameter l, the maximum rank of items currently considered for allocation. In the top level invocation of the procedure, $l = 1$.

The algorithm proceeds as follows:

1: **if** $U = \emptyset$ **then**
2: an allocation (Z_A, Z_B) has been found, return
3: **end if**
4: **for** all pairs $(i_A, i_B) : i_A \in H_A(l), i_B \in H_B(l), i_A \neq i_B$ **do**
5: allocate (i_A, i_B) and continue with $l + 1$

6: **end for**
7: **if** no such pair exists **then**
8: continue with $l + 1$
9: **end if**

In line 7, the algorithm checks whether any allocation can be made at a given level of the two rankings. For example, if both players share the same top item, no allocation is possible at level $l = 1$. In such a case, the algorithm continues with lower-ranked items. In the example, at least two allocations will be available at level $l = 2$.

3.3 Restricted SA algorithm (RS)

This algorithm was specifically defined for this study. Note that in line 5 of the OS algorithm, it can happen that neither player receives his or her top-ranked unallocated item. (Of course, these items will be allocated at a later stage.) The additional branches thus generated may or may not lead to identical allocations. The RS algorithm avoids this eventuality with a simpler procedure that can have at most two branches at each step:

1: **if** $top_A(U) \neq top_B(U)$ **then**
2: allocate $(top_A(U), top_B(U))$ and continue with $l + 1$
3: **else**
4: **if** $|H_A(l)| > 1$ **then**
5: allocate $(sb_A(H_A(l), top_B(H_B(l))$ and continue with $l + 1$
6: **end if**
7: **if** $|H_B(l)| > 1$ **then**
8: allocate $(top_A(H_A(l)), sb_B(H_B(l))$ and continue with $l + 1$
9: **end if**
10: **end if**

In lines 4 to 9, players who do not receive their top ranked unallocated item (because it is allocated to the opponent) receive their second best item in $H_m(l)$. Note that at this step, the algorithm will branch into two different paths only if there exists a second-best item with rank $r_m(sb_m(U)) \leq l$ for both players m. If this is not the case, only the player who has a second-best item at rank l or better will receive this second best item. Thus RS avoids allocating lower-ranked items as much as possible, which should increase the chances of finding a max-min allocation.

3.4 Singles-Doubles and Iterated Singles-Doubles algorithms (SD, IS)

These two algorithms were introduced by Brams et al. (2016). They classify items into two groups, *singles* and *doubles*. Define the max-min rank of any problem as

$$k = \max_{i \in S} \min_{m \in M} r_m(i) \tag{13}$$

Thus, every item is at rank k, or better, for at least one player. Items in $H_m(k)$ for only one player $m \in M$ are called singles; other items, which must be in both $H_A(k)$ and $H_B(k)$, are called doubles.

The algorithms proceed in two phases. In the first phase of SD, all singles are allocated to the player who most prefers them. This allocation is unique, so there is no branch in this phase. The allocation must be balanced, since if x singles are allocated to player A, there are $k - x$ doubles in $H_A(k)$. These doubles are also contained in $H_B(k)$, so there must also be x singles in $H_B(k)$.

The second phase processes all doubles. Note that when this phase begins, U is the set of doubles (since all singles are already allocated). This phase proceeds as follows:

1: **if** $U = \emptyset$ **then**
2: an allocation (Z_A, Z_B) has been found, return
3: **end if**
4: **if** $top_A(U) \neq top_B(U)$ **then**
5: allocate$(top_A(U), top_B(U))$ and continue
6: **end if**
7: **if** allocating $(top_A(U), sb_B(U))$ yields an envy-free partial allocation **then**
8: allocate$(top_A(U), sb_B(U))$ and continue
9: **end if**
10: **if** allocating $(sb_A(U), top_B(U))$ yields an envy-free partial allocation **then**
11: allocate$(sb_A(U), top_B(U))$ and continue
12: **end if**

Note that if neither the test in line 7 nor the test in line 10 indicates that an envy-free allocation is possible, then the algorithm terminates without finding an allocation. This will happen if, and only if, the given problem admits no envy-free allocation.

The IS algorithm differs from SD only in the first phase. After allocating all single items in a problem, the max-min item among the remaining items is determined and the first phase is repeated until no more singles can be identified (i.e. until the current max-min item is $last_m(U)$ for both players). Phase 2 is identical to SD.

In any problem for which an envy-free allocation exists, both SD and IS will generate allocations that are Pareto optimal, max-min and envy-free. If a problem has no envy-free solution, the SD and IS algorithms terminate without generating any allocation.

3.5 Modifications of SD and IS (S1 and I1)

The S1 and I1 algorithms are modifications of SD and IS developed for this study. They are designed to provide an allocation even in problems with no envy-free allocation. The first phases of S1 and I1 are identical to the first phases of SD and IS, respectively. If no envy-free allocation is available in lines 7 and 10 of the second phase of SD or IS, the algorithms proceed by allocating $(top_A(U), sb_B(U))$ and $(sb_A(U), top_B(U))$, and consequently generate

allocations that are not envy-free. However, this event occurs only if no envy-free allocation exists for the problem. For problems that do admit an envy-free allocation, S1 and I1 generate exactly the same allocations as SD and IS, respectively.

3.6 Bottom-Up algorithm (BU)

This algorithm was introduced by Brams and Taylor (1996). In contrast to the algorithms discussed so far, it considers players sequentially. Therefore, it will generate at most two allocations, depending on which player is considered first. The algorithm is extremely simple: when considering player m, it allocates item $last_m(U)$ to m's opponent $\overline{m}$. Despite its simplicity, this algorithm always generates allocations that are max-min.

3.7 Trump algorithm (TR)

This algorithm was developed by Pruhs and Woeginger (2012). It also considers players sequentially and thus generates up to two allocations when each of the two players is considered first. The TR algorithm proceeds as follows:

1: **for** all odd number $l \leq N$ **do**
2: **for** all players m **do**
3: **if** $H_m(l) = \emptyset$ **then**
4: Terminate, no envy free allocation exists
5: **end if**
6: allocate $last_{\overline{m}}(H_m(l))$ to player m
7: **end for**
8: **end for**

The TR algorithm generates only envy-free allocations and fails if no envy-free allocation exists. The allocations it generates are Pareto optimal, but not necessarily max-min.

4 Computational study

4.1 Research questions

The overall research question of this study is how well each of the algorithms outlined in section 3 performs with respect to the properties defined in section 2. Now we describe in more detail the specific questions we will address.

All of the algorithms we consider might generate multiple solutions to the same problem. Algorithms that process players' rankings in turn might produce different allocations depending on which player is considered first. Algorithms that consider rankings of both players simultaneously must make decisions when an item is contested, i.e., is the top-ranked unallocated item for both players. Contested items can lead to branching within the algorithm and thus generate multiple allocations. Our first research question is therefore:

RQ1: *How many allocations are generated by each algorithm on average, and in extreme cases? How often does an algorithm find a unique allocation?*

Good performance of an algorithm with respect to the properties we introduced can be measured in several ways. On the one hand, allocations that do not exhibit the desired properties should be avoided. An algorithm might produce several allocations for a given problem, but it should not generate any (or not many) that are not useful. On the other hand, it is important for an algorithm to find good allocations if they exist. This property is particularly relevant at the level of problems rather than of individual allocations: If for a problem one or more allocations with desired properties exist, but the algorithm fails to find any of them, that is a weakness of the algorithm.

Therefore, our second research question consists of three parts:
RQ2a: *What fraction of the allocations generated by each algorithm exhibit desirable properties?*
RQ2b: *What fraction of the allocations exhibiting desirable properties are found by each algorithm?*
RQ2c: *In what fraction of problems does each algorithm find at least one allocation with desirable properties, if such an allocation exists?*

Finally, we also want to study similarity of the results of different algorithms. The algorithms we analyze in this study are not very demanding from a computational point of view, so if some of them generate very different results, it could be useful to apply them all and then select the allocation that is in some sense "best." If, on the other hand, all algorithms tend to generate the same allocations most of the time, this approach would not be useful. Our third research question is therefore:
RQ3: *What fraction of the good allocations found by each algorithm are also found by other algorithms?*

4.2 Computational model

As outlined in the introduction, our computational study is exhaustive in the sense that we analyze all possible preference profiles for the problem sizes studied and all possible balanced allocations for each preference profile. Since items can, without loss of generality, be numbered according to the preferences of player A, the program generates all possible permutations of items as possible rankings for player B. We refer to each permutation as one problem.

For each problem, the program first generated all possible balanced allocations and assessed each one for the properties described in section 2. Thus, across all N-item problems, a total of $N!\binom{N}{N/2}$ allocations (i.e., 442,597,478,400 allocations for $N = 12$) were analyzed. Then, all eight algorithms were run on the problem and all allocations found by each algorithm were identified. Thus, at the end of this step, we have found for each balanced allocation the set of properties it possesses, and the set of algorithms that find it. These sets form the basis of our data analysis. To aggregate results across problems, we created a data matrix recording, for each possible subset of properties and each possible subset of algorithms, the number of allocations exhibiting the respective

properties that were found by the respective algorithms. By considering sets of properties and algorithms, rather then properties or algorithms individually, we are able to determine how many allocations are found by two or more algorithms, and how many allocations exhibit various combinations of properties. A similar data matrix was also created at the problem level, indicating the number of problems in which all algorithms in a set find at least one allocation having certain properties.

These two data matrices were created and saved for all problem sizes. For problems up to $N = 10$, additional information at the problem level was recorded to allow for more detailed analyses. For $N \leq 8$, information at the level of individual allocations was also recorded. All data generated can be obtained from the authors upon request.

The simulation was programmed in Pascal using the open source FreePascal compiler version 3.0 (`http://www.freepascal.org`). Running time for $N = 12$ items was about a week on a 3 GHz Intel i7 processor, and about ninety minutes for $N = 10$ on the same system.

5 Results

5.1 Numbers of allocations

Our first research question concerns the number of different allocations each algorithm generates for a problem. Upper bounds on these numbers are easy to obtain for most algorithms. BU and TR, for instance, consider the two players sequentially and never branch during execution, so they can generate at most two different allocations each. (Of course, they sometimes may generate the same allocation no matter which player is considered first.) Algorithms that consider only the top-ranked unassigned items in both players' rankings can assign a contested item to either player, and will therefore generate at most $2^{N/2}$ allocations, which occurs if they branch at each of the $N/2$ steps. Since items are allocated to different players at each branching point, all allocations resulting from different branches are different. Only OS can consider more than two items per player in a step, and thus might even branch more often, but different branches in OS can lead to the same allocation.

Example 1: Consider the application of OS to the problem in which B ranks the $N = 4$ items as $1 \succ 3 \succ 2 \succ 4$. (In all of our examples, we assume w.l.o.g. that items are numbered according to the preferences of player A, so the problem is specified by the ranking of player B.) In the first step, item 1 is contested. In the next step, A can receive either item 1 or item 2, and B either 1 or 3, producing 3 partial allocations; so the OS algorithm branches threefold at this point. In step 3, there is only one contested item in each path, and in step 4, both players can receive either of the remaining two unassigned items, leading again to two branches. Thus, the algorithm can take 6 paths in total. However, some allocations are achieved via two different paths. For example, if 1 is allocated to A and 3 is allocated to B in step 2, and then 2 is allocated

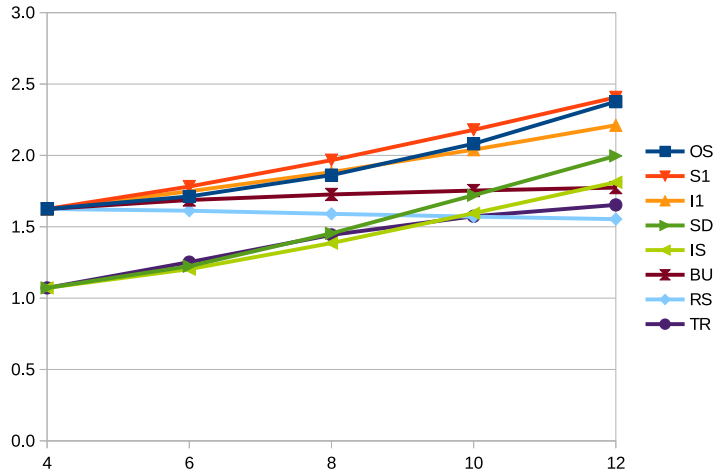

Figure 1: Average number of allocations generated

to A and 4 is allocated to B in step 4, the resulting allocation ($\{1, 2\}$ to A and $\{3, 4\}$ to B) is the same as the allocation that is obtained if in the second step 2 were allocated to A and 3 to B, and in the fourth step 1 to A and 4 to B. In this example, OS has six paths but only four allocations.

Figure 1 shows the average number of allocations generated by each algorithm for problems of different sizes. In this figure, we consider only problems in which an algorithm actually generates at least one allocation, so for algorithms that generate only EF allocations, problems for which no envy-free allocation exists are not included. TR frequently generates the same allocation regardless of which player is considered first; its average number of allocations barely exceeds one for small problems (1.071 for 4 items and 1.252 for 6 items). BU consistently generates one allocation in about a third of all problems (and two in the remainder); its average number of allocations remains close to 1.67 for all problem sizes.

It is also interesting to look at extreme cases—problems in which an algorithm produces just one allocation, or problems where the maximum number of allocations are generated. RS, I1 and S1 all reach the theoretical maximum of $2^{N/2}$ allocations for all problem sizes, while SD and IS both generate at most $2^{(N/2)-1}$ allocations. We conjecture that S1 and I1 branch in every step and thus generate $2^{N/2}$ allocations only in problems where preferences are so similar that no envy-free allocation exists. Furthermore, the maximum number of allocations found by OS in our data is also $2^{N/2}$.

Figure 2 shows the fraction of problems in which algorithms generate a unique allocation (again considering only problems in which the algorithm actually finds an allocation). RS differs from the other algorithms in that this number increases in problem size, while for all other algorithms, it decreases (apart

from a small increase for I1 and S1 between 4 and 6 items). The decrease is particularly strong for algorithms that generate only envy-free allocations.

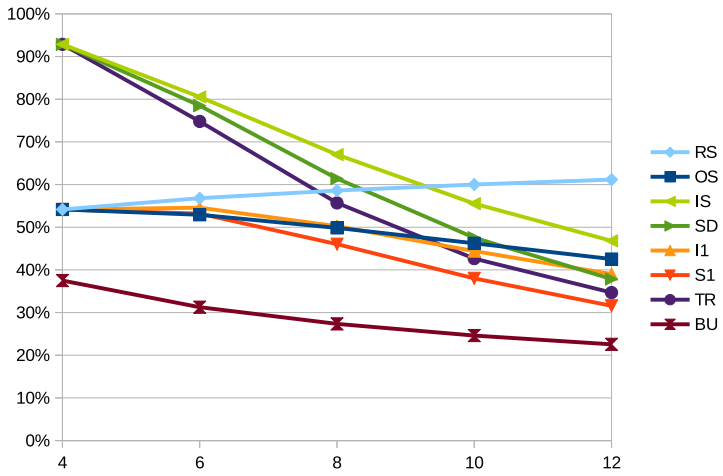

Figure 2: Fraction of problems in which a unique allocation is generated

5.2 Ordinal properties

Figure 3 shows how frequently the properties EF (envy-freeness), MM (max-min) and PO (Pareto optimality) occur among all possible balanced allocations for each problem size. Line "All" in this figure represents the fraction of allocations that exhibit all three properties. Envy-freeness seems to be the most critical of these properties, at least for problem sizes up to $N = 10$. Even for problems with 4 items, only about 10% of allocations are EF. In most cases, allocations that are EF also exhibit the other two properties; the line indicating the fraction of allocations with all properties is never far below the line for EF.

As explained in section 4, good performance of an algorithm with respect to a property or a set of properties on the one hand means that most of the allocations generated exhibit the desired properties, *and* that most allocations that exhibit the desired properties are actually found. We consider both questions in turn, first for ordinal properties and then for properties based on Borda counts. To simplify the presentation of results, we focus on allocations with all desired properties, rather than individual properties.

Table 1 addresses the first part of the question and shows the fraction of all allocations generated by each algorithm that exhibit all three properties. Since SD and IS can be shown to generate only EF and MM allocations, it is not surprising that all allocations found by these algorithms have all three properties (note that EF and MM seem to imply PO). Interestingly, for most other algorithms the fraction of "good" allocations increases with problem size. All start at the same level for problems with 4 items. As the number of items increases,

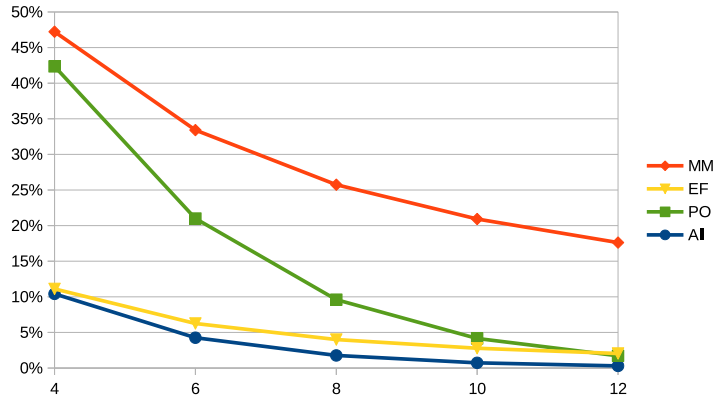

Figure 3: Fraction of allocations exhibiting the properties EF, MM and PO

Table 1: Fraction of generated allocations that are EF, MM, and PO (in %)

	Number of items				
Algorithm	4	6	8	10	12
RS	38.46	46.60	53.53	58.42	61.65
OS	38.46	45.17	49.58	51.38	51.21
SD	100.00	100.00	100.00	100.00	100.00
IS	100.00	100.00	100.00	100.00	100.00
S1	38.46	46.53	55.42	63.15	69.20
I1	38.46	46.62	55.15	62.52	68.34
BU	38.46	45.68	52.34	58.01	62.69
TR	100.00	92.14	85.69	84.77	85.99

OS falls somewhat behind the other algorithms. TR exhibits a particularly interesting pattern. Like the other two algorithms that guarantee envy-freeness, it starts out at 100% good allocations for problems with four items. The fraction of such allocations first decreases, but then stabilizes around 85%; thus, for larger problems, about 15% of the allocations generated by TR are not MM.

Since many algorithms generate similar fractions of "good" allocations, it might be thought that they all generate more or less the same allocations. This is not entirely the case. Figure 4 considers the composition of the sets of "good" allocations for problems with 12 items. At this problem size, exactly 1,323,482,551 balanced allocations are EF, MM and PO. We split this set into four groups: *exclusive* allocations, which are found only by the algorithm under consideration, *shared* allocations, which are found by at least one other algorithm, *missed* allocations, which are not found by the algorithm under consideration, but by some other algorithm, and finally the 213,241,844 (about 16% of total) allocations that were not found by any of the eight algorithms we studied, which are denoted as *All missed* in Figure 4. Since SD and S1, as well as IS and I1 always generate the same allocations in problems where an envy-free allocation exists, we treat each of these two pairs as one algorithm for this analysis. RS generates a subset of the allocations generated by OS. However, the results of the two algorithms do not necessarily coincide for problems where an envy-free allocation exists, so we analyze them separately, but nonetheless count allocations found by just these two algorithms as "exclusive". In total, SD (and thus also S1) finds the largest fraction of good allocations (about 60%). It also finds good allocations that none of the other algorithms finds (4.6% of all allocations). TR's fraction of exclusives, 5.6%, is even higher.

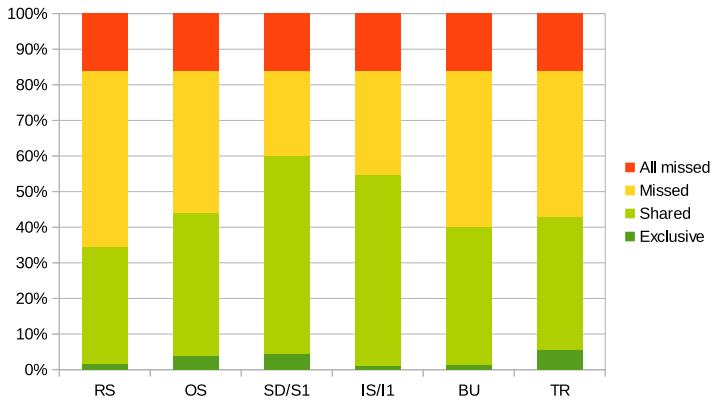

Figure 4: Structure of "good" allocations for problems with 12 items

In fact, BU and TR find exclusive allocations for problems with only $N = 8$ items, but not for smaller problems. At least for $N = 8$, there is no problem in which more than one of these algorithms generates an exclusive allocation. We now provide some examples of problems with $N = 8$ where there is an exclusive

solution.

Example 2: If B's ranking is $2 \succ 4 \succ 5 \succ 6 \succ 7 \succ 8 \succ 1 \succ 3$, OS is the only algorithm that finds the allocation $\{1, 3, 5, 6\}$ to A and $\{2, 4, 7, 8\}$ to B. Note that after the second round of any algorithm that processes the rankings top-down, all items 1, 2, 3, and 4 are already allocated, and among the remaining items, both players rank items 5 and 6 first and second. Thus, all other top-down algorithms would allocate 5 to one player and 6 to the other player, only OS allows both items to be allocated to the same player. It also generates the allocation$\{1, 3, 4, 7\}$ to A and $\{2, 5, 6, 8\}$ to B, but this allocation violates the MM property because it allocates item 7 to A.

Example 3: If B's ranking is $4 \succ 7 \succ 2 \succ 3 \succ 6 \succ 1 \succ 8 \succ 5$, BU is the only algorithm that allocates $\{1, 3, 5, 6\}$ to A and $\{2, 4, 7, 8\}$ to B. All algorithms that proceed top-down allocate item 2 to A.

Example 4: If B's ranking is $2 \succ 5 \succ 6 \succ 1 \succ 7 \succ 3 \succ 8 \succ 4$, only SD and S1 find the solution that allocates $\{1, 3, 4, 6\}$ to A. They first assign the singles 8 to B and 4 to A. After the first two steps of phase 2, only items 6 and 7 are left, which are ranked in that order by both players. SD now makes both possible assignments, which are both envy-free. The two pure top-down algorithms RS and OS would allocate item 6 to player B who ranks it higher. All the other algorithms allocate A's three worst items, 6, 7, and 8, to B.

Example 5: If B's ranking is $3 \succ 4 \succ 6 \succ 1 \succ 8 \succ 5 \succ 2 \succ 7$, only IS and I1 allocate $\{1, 2, 4, 7\}$ to A. In the first phase, items 7 and 2 are assigned to A and 8 and 6 to B. In the first step of the second phase, 1 is assigned to A and 3 to B, leaving items 4 and 5 unassigned. Both resulting allocations are envy-free, so IS generates them both. All other algorithms assign item 4 to B and 5 to A (and all other items in the same way as IS).

Example 6: If B's ranking is $2 \succ 3 \succ 8 \succ 5 \succ 6 \succ 1 \succ 7 \succ 4$, TR is the only algorithm that finds the allocation assigning $\{1, 3, 4, 5\}$ to A and $\{2, 6, 7, 8\}$ to B. This is the only EF allocation that assigns item 5 to A, the other two (more often found) "good" allocations assign 6 or 7 to A instead of 5. In this problem, BU also finds an allocation that assigns $\{1, 2, 4, 5\}$ to A, which assigns item 5 to A, but that allocation is not EF.

A similar analysis to the one in Figure 4 can also be performed at the level of problems. In our analysis, we found no problem of up to 12 items in which only one of the eight algorithms found an allocation fulfilling all three properties. There were also no problems in which none of the algorithms found such an allocation, if one existed. However, some algorithms failed to find any "good" allocation on some problems. Table 2 shows the fraction of problems in which algorithms failed to find any such allocation, although one exists. The last line of this table indicates the number of problems in which at least one "good" allocation exists.

Table 2: For each algorithm, fraction (in %) of problems in which at least one EF, MM and PO allocation exists but the algorithm found none

	Number of items				
	4	6	8	10	12
RS	0.00	4.92	8.42	11.10	13.26
OS	0.00	3.28	5.72	7.48	8.79
SD	0.00	0.00	0.00	0.00	0.00
IS	0.00	0.00	0.00	0.00	0.00
S1	0.00	0.00	0.00	0.00	0.00
I1	0.00	0.00	0.00	0.00	0.00
BU	0.00	1.64	2.30	2.22	1.84
TR	0.00	0.00	0.00	0.00	0.44
N. Problems	14	488	30,224	2,901,440	399,499,904

SD/IS always finds such an allocation, therefore, the same also holds for S1 and I1 (which differ only when no such solution exists). RS fails most often, and even in problems with only 6 items.

Example 7: If $N = 6$ and B's ranking is $3 \succ 4 \succ 5 \succ 6 \succ 1 \succ 2$, both RS and OS find only the allocation $\{1, 2, 5\}$ to A and $\{3, 4, 6\}$ to B, which violates max-min. The allocation of $\{1, 2, 4\}$ to A and $\{3, 5, 6\}$ to B, which fulfills all three criteria, is found by all other algorithms.

BU typically fails to find a "good" allocation because it does not consider envy-freeness.

Example 8: If $N = 6$ and B's ranking is $2 \succ 3 \succ 6 \succ 1 \succ 4 \succ 5$, BU generates two allocations, neither of which is envy-free: $\{1, 4, 5\}$ to A and $\{1, 2, 5\}$ to A. The allocation of $\{1, 3, 5\}$ to A, which fulfills all criteria, is found by all the other algorithms.

TR again exhibits an interesting pattern. This algorithm always finds a "good" allocation up to 10 items, but starts to miss some for 12 items. Since TR is guaranteed to provide an envy-free allocation, it fails only on the other properties.

Example 9: If $N = 12$ and B's ranking is $8 \succ 6 \succ 10 \succ 2 \succ 4 \succ 9 \succ 11 \succ 12 \succ 1 \succ 3 \succ 5 \succ 7$, then in the first 5 steps, TR assigns $\{1, 3, 5, 7, 9\}$ to A and $\{8, 10, 6, 11, 12\}$ to B (in that order), regardless of which player is considered first. Only in the last step does the sequence of players make a difference; either 2 is assigned to A and 4 to B, or vice versa. In any case, item 9 is assigned to A, which violates max-min. The max-min item is 12, ranked 8^{th} by B, but A receives his 9^{th} ranked item.

5.3 Borda properties

Four properties of allocations are defined in terms of Borda scores in section 2: Borda max-min (BM), Borda sum (BS), Borda proportionality (BE) and Borda Pareto optimality (BP). Since BS implies BP, an allocation that satisfies BM, BS and BE has all four properties.

We note that neither the maximum sum of Borda scores nor the max-min Borda score may be attained by any balanced allocation. We therefore consider an allocation to achieve BS or BM if it achieves the respective best value among all balanced allocations. Figure 5 illustrates the relationship between the constrained and unconstrained optima. For small problems, the max-min Borda allocation is typically balanced; for 6 items the fraction of balanced max-min solutions is 99.7%. With increasing problem size, this fraction drops, but still remains over 90% even for problems with 12 items. In contrast, the maximum sum of Borda scores is more often obtained at unbalanced allocations. It might be expected that a property related to efficiency is less often found in balanced allocations than a property related to fairness.

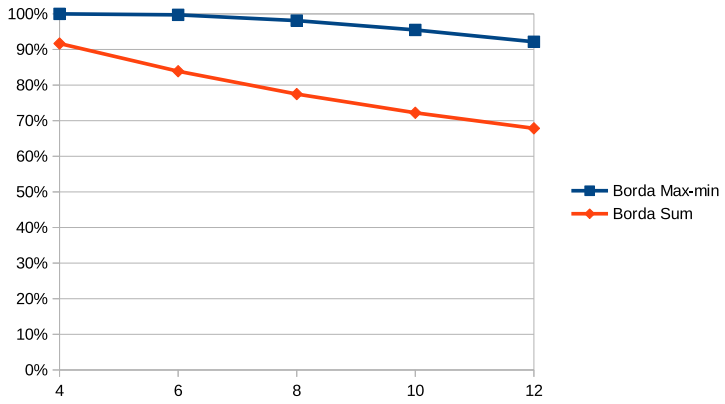

Figure 5: Fraction of problems in which a balanced allocation maximizes the sum of Borda scores and attains the max-min Borda score

Similar to Figure 3, Figure 6 shows the fraction of all allocations exhibiting all four Borda-related properties. In contrast to the ordinal properties, Borda proportionality occurs more often, it is the most frequent of the four properties, and it seems to be particularly easy to obtain if the number of items is divisible by four. This could be related to the fact that if preferences are identical and the number of items is a multiple of four, then balanced alternation can achieve equal Borda scores.

In analogy to Table 1, Table 3 shows the fraction of allocations generated by the algorithms that exhibit all desired Borda-related properties. In contrast to the properties defined directly from ordinal preferences, this share always decreases with increasing problem size. This phenomenon is not due to the specific design of the algorithms, which focuses on ordinal properties, but to

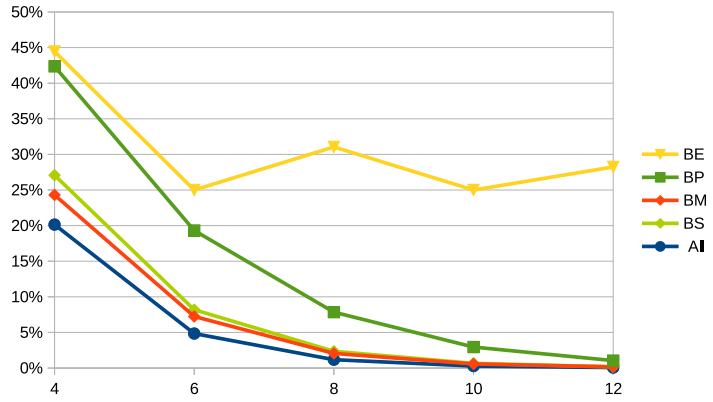

Figure 6: Fraction of allocations with Borda properties

Table 3: Fraction (in %) of generated allocations with all Borda properties

	Number of items				
	4	6	8	10	12
RS	74.36	57.80	47.06	38.62	32.05
OS	74.36	54.42	40.32	29.39	21.26
SD	100.00	77.22	52.83	36.41	25.88
IS	100.00	78.88	55.88	39.71	28.89
S1	74.36	52.14	37.95	27.93	20.96
I1	74.36	53.30	39.87	30.08	23.05
BU	69.23	50.12	38.34	30.56	24.91
TR	100.00	74.80	49.12	33.38	24.44

increasing rarity of Borda properties as problem size increases. Figure 7 shows the number of allocations that exhibit all ordinal as well as all Borda properties as a fraction of the number of allocations exhibiting all ordinal properties, and as a fraction of the allocations exhibiting all Borda properties, respectively. In problems with 4 items, all allocations exhibiting all ordinal properties also have all Borda properties, but not vice versa, only about half of the allocations having all Borda properties also have all ordinal properties. With increasing problem size, this relationship reverses. For problems with 12 items, only about 17% of the allocations having all ordinal properties also exhibit all Borda properties, while over 75% of the allocations having the Borda properties also exhibit the ordinal properties.

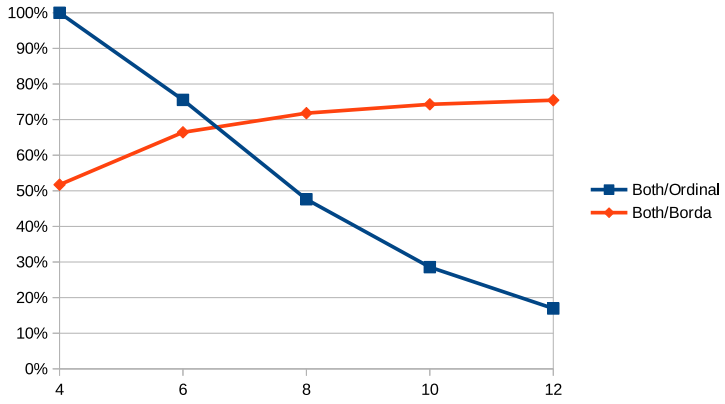

Figure 7: Overlap of ordinal and Borda properties

Table 4 performs a similar analysis as Figure 4 for allocations exhibiting all Borda properties. In contrast to the ordinal properties there are some (albeit few) problems in which only one algorithm found an allocation fulfilling all Borda properties (column "Exclusive"), and some cases (6.6%) in which no algorithm found such an allocation although one exists ("All missed"). We therefore perform this analysis in terms of problems, rather than in terms of individual allocations, as in Figure 4. Column "Missed " gives the number of problems in which an algorithm did not find any allocation fulfilling all Borda properties, although some other algorithm found at least one. To take into account that some algorithms generate subsets of the allocations generated by other algorithms, we again considered cases in which both RS and OS, both SD and S1, and both IS and I1 find an allocation as "exclusive". In interpreting these values, keep in mind that SD, IS and TR find allocations only if an envy-free allocation exists for the problem. Therefore, these algorithms will not find an allocation fulfilling all Borda properties for any problem that has no EF allocation. This fact could explain the relatively high number of missed problems for these algorithms. BU has a similarly high number of missed problems, but BU alone found an allocation exhibiting all Borda properties more often

than any other algorithm. Taken together, these values seem to indicate that allocations found by BU are of a different character from those found by other algorithms.

Table 4: Fraction of problems in which an allocation with all Borda properties was generated or missed, given such an allocation exists (in %, problems with N=12 items)

	Exclusive	Shared	Missed	All missed
RS (1)	1.82	80.87	10.71	6.60
OS (1)	1.91	81.91	9.58	6.60
SD (2)	0.22	71.45	21.73	6.60
IS (2)	0.05	72.57	20.78	6.60
S1 (2)	0.29	83.00	10.11	6.60
I1 (2)	0.06	84.12	9.21	6.60
BU	1.26	72.51	19.63	6.60
TR	0.44	56.06	36.90	6.60

(1) Since allocations generated by RS are also generated by OS, we count problems in which both algorithms find an allocation as "exclusive"

(2) Same for SD and S1 as well as IS and I1

Many allocations that exhibit all Borda properties also exhibit all ordinal properties, an observation that holds even in cases where such allocations are found exclusively by one algorithm. OS is the only algorithm for which this occurs for problems with as few as 8 items.

Example 10: If $N = 8$ and B's ranking is $3 \succ 4 \succ 5 \succ 6 \succ 7 \succ 8 \succ 1 \succ 2$, OS is the only algorithm to assign $\{1, 2, 5, 6\}$ to A. All other algorithms except RS find the allocation $\{1, 2, 4, 6\}$ to A, but that allocation fails on the Borda max-min criterion. Here player B's Borda score is only 21, while in the former allocation, both players receive a Borda score of 22, which is the max-min value. After having allocated items 1 and 2 in phase 1, SD and all its variants begin phase 2 with item 3 being contested. Item 3 must then be allocated to player B to achieve envy-freeness, which leads to the allocation of item 4 to A.

For the same problem size, TR exclusively finds some allocations that fulfill all ordinal as well as all Borda properties, but only in problems in the which other algorithms also find such allocations.

Example 11: If $N = 8$ and B's ranking is $5 \succ 8 \succ 2 \succ 3 \succ 4 \succ 6 \succ 1 \succ 7$, TR is the only algorithm that allocates $\{1, 2, 3, 7\}$ to A. All other algorithms allocate $\{1, 2, 4, 7\}$ to A; both of these allocations fulfill all ordinal and all Borda properties.

5.4 Similarities and differences of algorithms

Our last research question focused on the similarity of all allocations found by the different algorithms, as well as the allocations satisfying desirable properties.

Table 5: Fraction of allocations found by each algorithm also found by other algorithms (in %, problems with 12 items)

	Also found by ...							
Found by	RS	OS	SD	IS	S1	I1	BU	TR
RS	100.00	100.00	54.00	47.39	80.00	72.16	45.18	37.31
OS	65.37	100.00	41.63	35.76	61.43	54.49	33.36	28.73
SD	50.37	59.41	100.00	77.16	100.00	77.16	49.56	48.94
IS	48.73	56.25	85.03	100.00	85.03	100.00	64.69	53.19
S1	51.64	60.66	69.20	53.39	100.00	80.64	42.50	33.86
I1	50.70	58.57	58.11	68.34	87.77	100.00	54.04	36.35
BU	39.56	44.67	46.51	55.09	57.64	67.34	100.00	41.98
TR	42.04	49.51	59.11	58.29	59.11	58.29	54.02	100.00
Average	55.98	67.21	62.98	60.22	79.57	76.37	54.00	44.52
Size	0.835	1.278	0.896	0.813	1.294	1.189	0.954	0.741
Weighted	67.01	52.60	70.32	74.12	61.49	64.24	56.60	60.05

Table 5 shows the percentage of all allocations found by the algorithm indicated in the row that were also found by the algorithm indicated in the column. Note that RS is a restricted variant of OS, and S1 and I1 are generalizations of SD and IS; thus, all allocations found by RS were also found by OS, and all allocations found by SD or IS were also found by S1 or I1, respectively.

Line "Average" in Table 5 indicates the average of the values indicated in the column above, weighted by the number of allocations each algorithm generates (i.e. the sum of the number of co-occurrences of allocations, divided by the sum of the number of allocations found by each algorithm). On average, S1 and I1 find the largest fraction of allocations found by other algorithms, while TR exhibits a considerably lower overlap with the other algorithms. However, this average is somewhat misleading. Algorithms that find more allocations in total are more likely to find a larger fraction of the allocations found by other algorithm. To correct for this effect, we take into account the total number of allocations generated by each algorithm. The row labeled "Size" in Table 5 represents the ratio of the number of allocations generated by the algorithm named in the column to the average number of allocations generated by all algorithms. For example, S1 generates almost 30% more allocations than average. In the "Weighted" line, the averages are corrected by this factor, demonstrating that an allocation generated by SD and IS is most likely generated by some other algorithm as well. In contrast, OS and BU seem to produce allocations that are more often distinct from those of other algorithms.

The algorithms were specifically designed to find "good" allocations in terms of the ordinal properties EF, MM, and PO. It is therefore also interesting to

study the degree of overlap between the algorithms with reference to allocations exhibiting these three properties. Table 6 repeats the analysis of Table 5, restricted to allocations with all three ordinal properties. Taking into account

Table 6: Fraction of allocations satisfying EF, MM and PO found by each algorithm that are also found by other algorithms (in %, 12 items)

	Also found by ...							
Found by	RS	OS	SD	IS	S1	I1	BU	TR
RS	100.00	100.00	87.59	76.88	87.59	76.88	58.18	54.87
OS	78.70	100.00	81.30	69.84	81.30	69.84	50.94	49.81
SD	50.37	59.41	100.00	77.16	100.00	77.16	49.56	48.94
IS	48.73	56.25	85.03	100.00	85.03	100.00	64.69	53.19
S1	50.37	59.41	100.00	77.16	100.00	77.16	49.56	48.94
I1	48.73	56.25	85.03	100.00	85.03	100.00	64.69	53.19
BU	50.10	55.74	74.20	87.88	74.20	87.88	100.00	66.96
TR	44.33	51.13	68.74	67.79	68.74	67.79	62.82	100.00
All	56.80	65.40	86.55	82.76	86.55	82.76	61.34	58.19
Size	0.708	0.899	1.231	1.117	1.231	1.117	0.822	0.876
Weighted	80.25	72.72	70.32	74.12	70.32	74.12	74.62	66.42

that it finds comparatively few "good" solutions, the likelihood that an RS allocation is also found by other algorithms is the greatest. On the other hand, IS and I1 find many "good" solutions, but they are to a large extent similar to what other algorithms generate. As noted earlier, TR seems to be the most distinct of the algorithms studied.

6 Discussion, conclusions, and future research

We have presented an exhaustive computational study that considered all possible balanced allocations of up to 12 items to two players, analyzed which of these allocations have desirable properties, and assessed the extent to which allocations with desirable properties can be found by eight algorithms. In general, we conclude that the algorithms we assessed are quite good at finding the "needle in the haystack": Although allocations exhibiting specific properties like envy-freeness, max-min, and Pareto optimality, let alone combinations of these properties (or similar properties defined on Borda scores), are quite rare, the algorithms typically find most of them. As well, our analysis identified some marked differences among algorithms. Of particular interest from an application point of view is whether an algorithm will find a "good" allocation (satisfying EF, MM and PO) if one exists. SD and IS will never fail in this respect, and the same necessarily holds for their modifications S1 and I1, which will produce exactly the same allocation if a good one exists. In contrast, other algorithms might fail in more than 5% of problems.

The situation is less encouraging with respect to properties formulated in

terms of Borda counts, where we find that, for more than 5% of problems where a good allocation exists, none of the algorithms finds any allocation with these properties. This is not entirely surprising, in that achieving Borda-related properties was not a goal when the algorithms were designed. However, we agree with Darmann and Klamler (2016) that these properties provide valuable measures of the quality of allocations. On the one hand, Borda scores are a conceptually simple way of evaluating bundles of items and require only ordinal information; on the other hand, they constitute a complete relation on bundles, albeit with some indifferences. We consider the latter point to be particularly important in reference to properties defined by the absence of certain elements in a relation (such as PO, which arises when there is no dominating allocation). For these properties, using an incomplete relation to compare bundles of items will result in many allocations exhibiting the properties, which might give an overly optimistic picture of the actual situation (assuming that actual preferences are in fact complete).

Another interesting result of our study is the comparatively large degree of overlap among results of the algorithms. As Table 6 shows, many of the "good" allocations (fulfilling all properties defined using ordinal preferences) are found not just by one, but by several of the algorithms. In particular, SD and IS seem to offer superior performance in that they find most of the good allocations found by other algorithms. However, some algorithms, in particular TR, contribute an additional set of allocations that is somewhat "orthogonal" to SD and IS. This observation suggests the strategy of applying several algorithms in parallel in order to obtain a broad spectrum of possible allocations that achieve desirable properties. (Table 6 provides a clear indication which algorithms should be combined.) Of course, increasing the number of allocations that are generated also complicates the problem of choosing among them. Although they all satisfy fairness criteria like envy-freeness, they are generally not equivalent for either player, so this final choice problem is far from trivial.

Furthermore, by considering all possible problems and all possible balanced allocations, our exhaustive study also enabled us to identify specific problems in which algorithms fail, and particular allocations that cannot be found. This information might help developers of future algorithms to avoid these pitfalls.

Although our computational study has provided several interesting insights, it certainly is not without limitations. Computational methods always raise questions of generalizability. While we have provided a complete analysis of all problems of the sizes we studied (thereby, for example, verifying theoretical results about these problems such as Theorem 6 of Darmann and Klamler (2016)), our results do not easily generalize to larger problem sizes. Several of our observations do not even exhibit a clear trend as the number of items increases, and it is a mystery how these results evolve as problem size increases. Computational studies like ours can complement theoretical results in a meaningful way, and can provide stylized facts that inspire further theoretical analyses as well as the development of new algorithms, but they cannot replace them.

References

Bansal, N. and Sviridenko, M. (2006). The Santa Claus problem. In Kleinberg, J., editor, *STOC'06 Symposium on Theory of Computing*, pages 31–40, Seattle, WA. ACM.

Bertsimas, D., Farias, V. F., and Trichakis, N. (2012). On the efficiency-fairness trade-off. *Management Science*, 58(12):2234–2250.

Bouveret, S., Chevaleyre, Y., and Maudet, N. (2016). Fair allocation of indivisible goods. In Brandt, F., Conitzer, V., Endriss, U., Lang, J., and Procaccia, A. D., editors, *Handbook of Computational Social Choice*, pages 284–310. Cambridge University Press, New York.

Bouveret, S. and Lemaître, M. (2016). Characterizing conflicts in fair division of indivisible goods using a scale of criteria. *Autonomous Agents and Multi-Agent Systems*, 30(2):259–290.

Brams, S. J., Kilgour, D. M., and Klamler, C. (2012). The undercut procedure: An algorithm for the envy-free division of indivisible items. *Social Choice and Welfare*, 39:615–631.

Brams, S. J., Kilgour, D. M., and Klamler, C. (2015). How to divide things fairly. *Mathematics Magazine*, 88(5):338–348.

Brams, S. J., Kilgour, D. M., and Klamler, C. (2016). Maximin envy-free division of indivisible items. *Mathematics Magazine*, forthcoming.

Brams, S. J. and Taylor, A. D. (1996). *Fair Division: From Cake-Cutting to Dispute Resolution*. Cambridge University Press, New York.

Darmann, A. and Klamler, C. (2016). Proportional Borda allocations. *Social Choice and Welfare*, 47(3):543–558.

Klamler, C. (2010). Fair division. In Kilgour, D. M. and Eden, C., editors, *Handbook of Group Decision and Negotiation*, pages 183–202. Springer, Dordrecht.

Pruhs, K. and Woeginger, G. J. (2012). Divorcing made easy. In Kranakis, E., Krizanc, D., and Luccio, F., editors, *FUN 2012*, Lecture Notes in Computer Science, pages 305–314. Springer, Berlin.

Thomson, W. (2016). Introduction to the theory of fair allocation. In Brandt, F., Conitzer, V., Endriss, U., Lang, J., and Procaccia, A. D., editors, *Handbook of Computational Social Choice*, pages 262–283. Cambridge University Press, New York.

Young, H. P. (1994). *Equity: In Theory and Practice*. Princeton University Press, Princeton, New Jersey.