Chapter 3

Outcome Optimization

The more alternatives, the more difficult the choice.

Abbé D'Allanival

Perhaps surprisingly, expressive preference aggregation has only recently started to be adopted in practice (mostly in the form of combinatorial auctions and combinatorial reverse auctions). One of the reasons for this is that in expressive preference aggregation protocols, the problem of choosing the best outcome once the preferences have been collected is typically computationally hard. For instance, in combinatorial auctions, the winner determination problem is NP-complete [Rothkopf *et al.*, 1998] (even to approximate [Sandholm, 2002a]). The computational tools for solving such outcome selection problems have only recently reached the maturity necessary to make some of these protocols feasible in practice [Sandholm *et al.*, 2006; Sandholm, 2006].

This chapter will discuss the outcome optimization problem for the domains discussed in Chapter 2. While computing the aggregated ranking is easy under most voting rules, there are some voting rules for which this is not the case [Bartholdi *et al.*, 1989b; Hemaspaandra *et al.*, 1997; Cohen *et al.*, 1999; Dwork *et al.*, 2001; Rothe *et al.*, 2003; Davenport and Kalagnanam, 2004; Ailon *et al.*, 2005], including the Kemeny and Slater rules. In Section 3.1, we introduce a powerful preprocessing technique for computing optimal rankings according to the Slater rule [Conitzer, 2006]. In Section 3.2, we give new conditions under which the winner determination problem in combinatorial auctions can be solved efficiently [Conitzer *et al.*, 2004]. In Section 3.3, we study computing optimal outcomes under the framework for negotiating over donations introduced in Section 2.3 [Conitzer and Sandholm, 2004e]. Finally, in Section 3.4 we study computing optimal outcomes under the framework for negotiating in settings with externalities introduced in Section 2.4 [Conitzer and Sandholm, 2005d].

3.1 A preprocessing technique for computing Slater rankings

This section describes work done at IBM Research, with guidance and advice from Andrew Davenport and Jayant Kalagnanam.

In voting, we would like our aggregate ranking to be consistent with the outcome of each pair-

wise election: if more voters prefer a to b than vice versa, the aggregate ranking should rank a ahead of b. Unfortunately, Condorcet cycles can prevent us from being able to achieve this consistency for all pairs of candidates. The *Slater* voting rule is arguably the most straightforward resolution to this problem: it simply chooses a ranking of the candidates that is inconsistent with the outcomes of as few pairwise elections as possible. Unfortunately, as we will discuss later in this section, computing a Slater ranking is NP-hard. This in stark contrast to the computational ease with which most voting rules can be executed (most rules require only a simple computation of a score for each candidate, or perhaps one score per candidate for every round of the rule). An exception is the related Kemeny rule, which is also NP-hard to compute (in fact, as we will show later in this section, this hardness is implied by the Slater rule's hardness). The hardness of computing Slater ranking suggests using a tree search-based algorithm to compute Slater rankings.[1]

In this section, we introduce a *preprocessing* technique that can reduce the size of an instance of the Slater problem before the search is started. We say that a subset of the candidates consists of *similar candidates* if for every candidate outside of the subset, all candidates inside the subset achieve the same result in the pairwise election against that candidate. Given a set of similar candidates, we can recursively solve the Slater problem for that subset, and for the original set with the entire subset replaced by a single candidate, to obtain a solution to the original Slater problem. In addition, we also make the following contributions:

- We show that if the results of the pairwise elections have a particular hierarchical structure, the preprocessing technique is sufficient to solve the Slater problem in linear time.
- For the general case, we give a polynomial-time algorithm for *finding* a set of similar candidates (if it exists). This algorithm is based on satisfiability techniques.
- We exhibit the power of the preprocessing technique experimentally.
- We use the concept of a set of similar candidates to give the first straightforward reduction (that is, not a randomized reduction or a derandomization thereof) showing that the Slater problem is NP-hard in the absence of pairwise ties.

3.1.1 Definitions

Recall that the Slater rule is defined as follows: find a ranking $\succ$ on the candidates that minimizes the number of ordered pairs (a, b) such that $a \succ b$ but b defeats a in their pairwise election. (Equivalently, we want to maximize the number of pairs of candidates for which $\succ$ *is* consistent with the result of the pairwise election—we will refer to this number as the *Slater score*.) We will refer to the problem of computing a Slater ranking as the Slater problem. An instance of the Slater problem can be represented by a "pairwise election" graph whose vertices are the candidates, and which has a directed edge from a to b if and only if a defeats b in their pairwise election. The goal, then, is to minimize the number of edges that must be flipped in order to make the graph acyclic.

Most elections do not have any ties in pairwise elections. For example, if the number of votes is odd, there is no possibility of a pairwise tie. (We note that in many real-world elections, the number

[1] Another approach is to look for rankings that are *approximately* optimal in the Slater sense [Ailon *et al.*, 2005; Coppersmith *et al.*, 2006]. Of course, this is not entirely satisfactory as it is effectively changing the voting rule.

of voters is intentionally made odd to prevent ties.) Hence, we will restrict our attention to elections without pairwise ties (in which case the pairwise election graph becomes a tournament graph). For our positive results, this is merely for simplicity—they can easily be extended to deal with ties as well. Our one negative result, the NP-hardness of computing Slater rankings, is made stronger by this restriction (in fact, without the restriction the hardness has effectively been known for a long time).

3.1.2 Sets of similar candidates

We are now ready to give the formal definition of a set of similar candidates.

Definition 9 *We say that a subset $S \subseteq C$ consists of* similar candidates *if for any $s_1, s_2 \in S$, for any $c \in C - S$, $s_1 \to c$ if and only if $s_2 \to c$ (and hence $c \to s_1$ if and only if $c \to s_2$).*

We emphasize that in this definition, it is *not* required that *every* vote prefers s_1 to c if and only if that vote prefers s_2 to c. Rather, the condition only needs to hold on the aggregated pairwise election graph, and hence it is robust to a few voters who do not perceive the candidates as similar.

There are a few trivial sets of similar candidates: 1. the set of all candidates, and 2. any set of at most one candidate. We will be interested in nontrivial sets of similar candidates, because, as will become clear shortly, the trivial sets have no algorithmic use.

The following is the key observation of this section:

Theorem 1 *If S consists of similar candidates, then there exists a Slater ranking $\succ$ in which the candidates in S form a* (contiguous) block *(that is, there do not exist $s_1, s_2 \in S$ and $c \in C - S$ such that $s_1 \succ c \succ s_2$).*

Proof: Consider any ranking $\succ_1$ of the candidates in which the candidates in S are split into $k > 1$ blocks; we will show how to transform this ranking into another ranking $\succ_2$ with the properties that:

- the candidates in S are split into $k - 1$ blocks in $\succ_2$, and
- the Slater score of $\succ_2$ is at least as high as that of $\succ_1$.

By applying this transformation repeatedly, we can transform the original ranking into an ranking in which the candidates in S form a single block, and that has at least as high a Slater score as the original ranking.

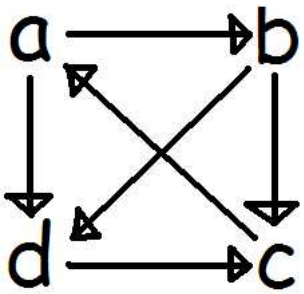
Consider a subsequence of $\succ_1$ consisting of two blocks of candidates in S, $\{s_i^1\}$ and $\{s_i^2\}$, and a block of candidates in $C - S$ that divides them, $\{c_i\}$: $s_1^1 \succ_1 s_2^1 \succ_1 \ldots \succ_1 s_{l_1}^1 \succ_1 c_1 \succ_1 c_2 \succ_1 \ldots \succ_1 c_l \succ_1 s_1^2 \succ_1 s_2^2 \succ_1 \ldots \succ_1 s_{l_2}^2$. Because S consists of similar candidates, a given candidate c_i has the same relationship in the pairwise election graph to *every* s_i^j. Hence, one of the following two cases must apply:

1. For at least half of the candidates c_i, for every s_i^j, $c_i \to s_i^j$
2. For at least half of the candidates c_i, for every s_i^j, $s_i^j \to c_i$.

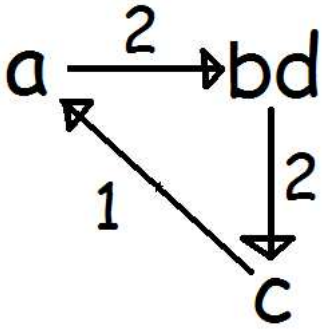
In case 1, we can replace the subsequence by the subsequence $c_1 \succ_2 c_2 \succ_2 \ldots \succ_2 c_l \succ_2 s_1^1 \succ_2 s_2^1 \succ_2 \ldots \succ_2 s_{l_1}^1 \succ_1 s_1^2 \succ_2 s_2^2 \succ_2 \ldots \succ_2 s_{l_2}^2$ to join the blocks without any loss to the Slater score of the ranking. Similarly, in case 2, we can replace the subsequence by the subsequence $s_1^1 \succ_2 s_2^1 \succ_2 \ldots \succ_2 s_{l_1}^1 \succ_1 s_1^2 \succ_2 s_2^2 \succ_2 \ldots \succ_2 s_{l_2}^2 \succ_2 c_1 \succ_2 c_2 \succ_2 \ldots \succ_2 c_l$ to join the blocks without any loss to the Slater score of the ranking. ■

Hence, if we know that S consists of similar candidates, then when we try to compute a Slater ranking, we can without loss of generality restrict our attention to rankings in which all the candidates in S form a (contiguous) block. The optimal internal ranking of the candidates in S within the block is independent of the rest of the ranking, and can be computed recursively.[2] Because of this, we can think of S as a single "super-candidate" with weight $|S|$. Ranking S above a candidate c such that $s \to c$ for all $s \in S$, or below a candidate c such that $c \to s$ for all $s \in S$, will increase the Slater score by $|S|$.

Consider the following pairwise election graph:

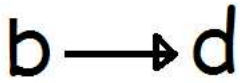


In this graph, $\{b, d\}$ is a set of similar candidates. Thus, we recursively solve the instance in which b and d are aggregated into a single candidate:



Some of the edges now represent multiple edges in the original graph; this is indicated by the weights on these edges. It is easy to see that the optimal Slater ranking for this graph is $a \succ bd \succ c$. In addition, we need to solve the Slater problem internally for the set of similar candidates:

[2]Note that if S is a trivial set of similar candidates, there is little use to this: if it is a set of at most one candidate, then the statement that that candidate will form a block by itself is vacuous, and if it is the set of all candidates, we need to recurse on the set of all candidates.



The optimal Slater ranking for this graph is (of course) $b \succ d$. So the solution to the original problem is $a \succ b \succ d \succ c$.

It is possible to have multiple disjoint sets S_i, each consisting of similar candidates. In this case, we can aggregate each one of them into a single super-candidate. The following lemma will clarify how to compute the Slater scores for such pairs of super-candidates:

Lemma 1 *If S_1 and S_2 are disjoint sets of similar candidates, then for any $s_1, s_1' \in S_1$ and any $s_2, s_2' \in S_2$, $s_1 \to s_2$ if and only if $s_1' \to s_2'$. (That is, the same relationship holds in the pairwise election graph for any pair of candidates in $S_1 \times S_2$.) Hence, ranking super-candidate S_1 above super-candidate S_2 such that $s_1 \to s_2$ for all $s_1 \in S_1, s_2 \in S_2$, or below a super-candidate S_2 such that $s_2 \to s_1$ for all $s_1 \in S_1, s_2 \in S_2$, will increase the Slater score by $|S_1| \cdot |S_2|$.*

Proof: Because S_1 consists of similar candidates, $s_1 \to s_2$ if and only if $s_1' \to s_2$. And, because S_2 consists of similar candidates, $s_1' \to s_2$ if and only if $s_1' \to s_2'$. ■

Similar sets that overlap cannot be simultaneously turned into super-candidates. However, the following lemma shows that turning one of them into a super-candidate will (in a sense) preserve the structure of the other set: after aggregating one of the sets into a super-candidate, the other set will, in a sense, coincide with the union of the two sets, and we now show that this union must consist of similar candidates.

Lemma 2 *If S_1 and S_2 each consist of similar candidates, and $S_1 \cap S_2$ is nonempty, then $S_1 \cup S_2$ consists of similar candidates.*

Proof: Let $s \in S_1 \cap S_2$. For any $s', s'' \in S_1 \cup S_2$ and any $c \in C - (S_1 \cup S_2)$, we have that $s' \to c$ if and only if $s \to c$, and $s \to c$ if and only if $s'' \to c$. ■

3.1.3 Hierarchical pairwise election graphs can be solved in linear time

In this subsection, we show that if the pairwise election graph has a certain hierarchical structure, then the Slater problem can be solved efficiently using the techniques from the previous subsection.

Definition 10 *A* valid candidate tree *is a tree with the following properties:*

- *The leaves are each labeled with a candidate, with each candidate appearing exactly once.*
- *For every internal vertex v, there is a tournament graph $\to_v$ over its children such that for any two distinct children $w_1 \neq w_2$ of v, for any descendants d_1 of w_1 and d_2 of w_2, $d_1 \to d_2$ if and only if $w_1 \to_v w_2$.*

Put alternatively, to find out the direction of the edge between any two candidates in the pairwise election graph, we can simply compare the vertices directly below their least common ancestor. There is always a trivial valid candidate tree, which simply connects every candidate directly to the root node R and uses the pairwise election graph $\rightarrow$ as the graph $\rightarrow_R$. This tree does not give us any insight. Instead, we will be interested in trees whose vertices have small degree (that is, each vertex has only a few children).

Figure 3.1 shows an example candidate tree, and Figure 3.2 shows the corresponding graph of pairwise elections.

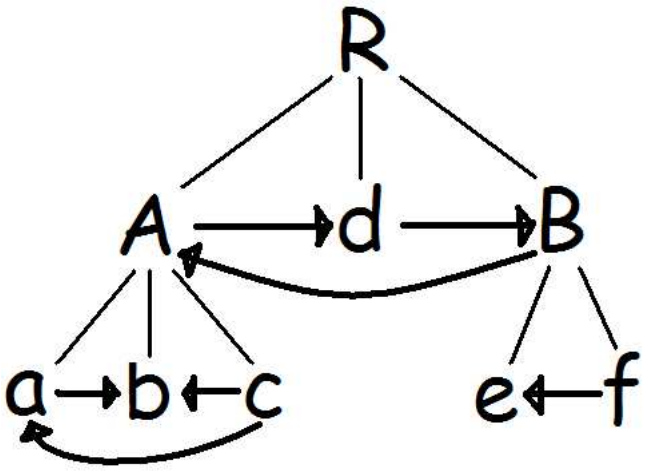


Figure 3.1: A valid candidate tree.

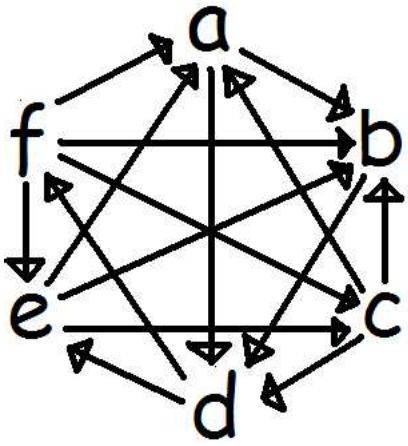


Figure 3.2: The pairwise election graph corresponding to the valid candidate tree.

The following observation will allow us to use the structure of the tree to solve the Slater problem efficiently:

Lemma 3 *For any vertex v in a valid candidate tree, the set D_v of candidates that are descendants*

of v constitutes a set of similar candidates.

Proof: For any $d_1, d_2 \in D_v$ and $c \in C - D_v$, the least common ancestor of d_1 and c, or of d_2 and c, must be a (strict) ancestor of v. Hence, this least common ancestor must be the same in both cases, and moreover, so must the child of that ancestor from which d_1 and d_2 (and v) descend. Hence $d_1 \to c$ if and only if $d_2 \to c$. ■

Hence, we can solve the Slater problem using the following very simple algorithm:

1. For every child of the root R, generate a super-candidate with weight equal to the number of candidates that descend from it.

2. Solve the Slater problem for the graph $\to_R$ over these super-candidates (using any algorithm).

3. Solve the Slater problem recursively for each subtree rooted at a child of the root R.

4. Order the candidates, first according to the ranking of the super-candidates that they are in, and then according to the recursive solutions.

Step 2 may be computationally expensive, depending on the number of super-candidates. However, if the degree of each vertex is small, then so is the number of super-candidates in this step. In particular, if the degree is bounded by a constant, then step 2 can be performed in constant time, and the running time of the entire algorithm is linear.

The algorithm produces the Slater ranking $f \succ e \succ c \succ a \succ b \succ d$ on the example given above.

3.1.4 An algorithm for detecting sets of similar candidates

In general, we do not know in advance whether there is a nontrivial set of similar candidates in a pairwise election graph. Rather, we need an algorithm that will take as input a pairwise election graph, and discover a nontrivial set of similar candidates if it exists. In this subsection, we present such an algorithm. The algorithm relies on transforming the problem of detecting a set of similar candidates into a Horn satisfiability problem.

Specifically, for every candidate c we generate a variable $In(c)$ which indicates whether the candidate is in the set of similar candidates. Then, for every ordered triplet of candidates $c_1, c_2, c_3 \in C$, if either $c_1 \to c_3$ and $c_3 \to c_2$, or $c_2 \to c_3$ and $c_3 \to c_1$, then we generate the clause $In(c_1) \wedge In(c_2) \Rightarrow In(c_3)$ (or, equivalently, $(\neg In(c_1) \vee \neg In(c_2) \vee In(c_3))$.

The instance described two subsections earlier produces the following clauses: $In(a) \wedge In(b) \Rightarrow In(c)$, $In(a) \wedge In(c) \Rightarrow In(b) \wedge In(d)$, $In(a) \wedge In(d) \Rightarrow In(b) \wedge In(c)$, $In(b) \wedge In(c) \Rightarrow In(a) \wedge In(d)$, $In(c) \wedge In(d) \Rightarrow In(a)$.

Theorem 2 *A setting of the variables $In(c)$ satisfies all the clauses if and only if $S = \{c \in C : In(c) =$*true*$\}$ consists of similar candidates.*

Proof: First, suppose that the setting satisfies all the clauses. For any $s_1, s_2 \in S$ and $c \in C - S$, if there were a clause $In(s_1) \wedge In(s_2) \Rightarrow In(c)$, it would not be satisfied. It follows that this clause was not generated, and hence either $s_1 \rightarrow c$ and $s_2 \rightarrow c$, or $c \rightarrow s_1$ and $c \rightarrow s_2$. Hence S consists of similar candidates.

Next, suppose that the setting does not satisfy all the clauses. Then, there must be some unsatisfied clause $In(c_1) \wedge In(c_2) \Rightarrow In(c_3)$, which means that $c_1, c_2 \in S$ and $c_3 \notin S$. Because the clause was generated, either $c_1 \rightarrow c_3$ and $c_3 \rightarrow c_2$, or $c_2 \rightarrow c_3$ and $c_3 \rightarrow c_1$, and hence S does not consist of similar candidates. ■

There are some settings of the variables $In(c)$ that always satisfy all the clauses: setting everything to *true*, and setting at most one variable to *true*. These settings correspond exactly to the trivial sets of similar candidates discussed earlier. Hence, our goal is to find a satisfying setting of the variables in which at least two, but not all, variables are set to *true*. In the example above, the only such solution is to set $In(b)$ and $In(d)$ to *true* and $In(a)$ and $In(c)$ to *false*, corresponding to the set of similar candidates that we used earlier in this section. Finding a nontrivial solution can be done in polynomial time with the following simple algorithm: for a given pair of candidates $c_1, c_2 \in C$, set $In(c_1)$ and $In(c_2)$ to *true*, and then follow the implications $\Rightarrow$ in the clauses. If this process terminates without setting all the $In(c)$ variables to *true*, we have found a nontrivial set of similar candidates. Otherwise, restart with a different pair of candidates, until we have tried every pair of candidates. The algorithm can be improved by keeping track of the initial pairs of candidates for which we have failed to find a similar set, so that when another initial pair leads to one of these pairs being set to *true*, we can fail immediately and continue to the next pair.

When we use this algorithm for finding similar sets to help us compute a Slater ranking, after finding a similar set, we need to compute a Slater ranking both on the instance consisting of the similar set only, and on the reduced version of the original instance where the similar set has been replaced by a single super-candidate. Thus, we will be interested in finding similar sets on these instances as well. It is natural to ask whether some of the computation that we did to find a similar set in the original instance can be reused to find similar sets in the two new instances. It turns out that, in the second new instance, this is indeed possible:

Lemma 4 *Suppose that, in the process of detecting a similar set, we failed with the pair of initial candidates $c_1, c_2 \in C$, before discovering that $S \subseteq C$ is a similar set. Then, in the reduced instance where S is replaced by a single super-candidate c_S,*

1. *we will also fail on initial pair c_1, c_2 if $c_1, c_2 \notin S$;*
2. *we will also fail on initial pair c_1, c_S if $c_1 \notin S, c_2 \in S$.*

Proof: Suppose c_1, c_2 (or, in the second case, c_1, c_S) belong to a nontrivial set S' of similar candidates in the reduced instance. Then, consider the same set in the original instance (if $c_S \in S'$, replace it with all members of S); call this set S''. S'' does not include all candidates because S' does not include all candidates in the reduced instance. Moreover, S'' is a set of similar candidates, for the following reasons. Take any $s_1, s_2 \in S''$ and $c \in C - S''$; we must show that $s_1 \rightarrow c$ if and only if $s_2 \rightarrow c$. If $s_1 \notin S$ and $s_2 \notin S$, then this follows from the fact that S' consists of similar candidates (even if $c \in S$, because in that case $s_i \rightarrow c$ if and only if $s_i \rightarrow c_S$ in the reduced

instance). If exactly one of s_1 and s_2 is in S (without loss of generality, say $s_1 \in S$), then it must be that $c_S \in S' \Leftrightarrow S \subseteq S''$, so that $c \notin S$. Hence, $s_1 \rightarrow c$ if and only if $c_s \rightarrow c$, and because S' consists of similar candidates this is true if and only if $s_2 \rightarrow c$. Finally, if both s_1 and s_2 are in S, then $c \notin S$ because $c_S \in S' \Leftrightarrow S \subseteq S''$, hence $s_1 \rightarrow c$ if and only if $s_2 \rightarrow c$ because S consists of similar candidates. But if S'' consists of similar candidates, then we would not have failed on the initial pair c_1, c_2 in the original instance. Hence we have the desired contradiction. ■

For the first new instance consisting of S only, such reuse of computation is not possible, because we cannot have failed on a pair of candidates within S (since they were in fact in a similar set). We only know that we will fail on the pair starting with which we found S, because this pair will lead to all candidates in S being included in the similar set.

3.1.5 Experimental results

In this subsection, we experimentally evaluate the use of the techniques described above as a preprocessing technique that serves to reduce the sizes of the instances before a search algorithm is called. We compare two algorithms: a straightforward search technique, and the preprocessing technique combined with the same search technique. (Instead of the straightforward search technique, we could also have used a more sophisticated algorithm, *e.g.* a commercial solver such as CPLEX. The goal here, however, is not to show that our technique by itself outperforms any given algorithm, but rather it is to show that using the preprocessing technique can reduce a given algorithm's computation time. Moreover, if adding the preprocessing technique to a straightforward search algorithm already produces a fast algorithm, that is more impressive than having to add the technique to a solver such as CPLEX to obtain a fast algorithm.) The straightforward search technique decides, at every search node, whether a given edge in the graph should be consistent or inconsistent with the final ranking, and then propagates the effect of this decision to other edges (*e.g.* by transitivity, if it has been decided that edges (a, b) and (b, c) will both be consistent with the final ranking, then by transitivity so must the edge (b, c)). There is no sophisticated variable ordering heuristic: we simply choose the first edge that has not yet been set. As an admissible pruning heuristic, we use the total number of edges for which it has been decided that the final ranking will be inconsistent with them.

The preprocessing technique uses the algorithm described in the previous subsection to search for a set of similar candidates. If it finds one, it recursively solves the subproblems; otherwise, the search algorithm is called to solve the remaining irreducible problem instance. Each data point in the experiments is an average of 30 instances (the same instances for both algorithms).

In the first set of experiments, instances are generated as follows. Every candidate and every voter draws a random position in $[0, 1]$ (this can be thought of as their stance on one issue) and voters rank candidates by proximity to their own position. The results are in Figure 3.3:

On these instances, even straightforward search scales reasonably well, but when the preprocessing technique is added, all the instances solve immediately. This is not surprising: the voters' preferences in this domain are *single-peaked*, and it is well-known that for single-peaked preferences, there are no cycles in the pairwise election graph (*e.g.* [Mas-Colell *et al.*, 1995]), so that the final ranking can be read off directly from the graph. Given this, any k candidates in contiguous positions in the final ranking always form a set of similar candidates, so that the preprocessing technique can solve the instances entirely. (No time is spent in search after the preprocessing technique.)

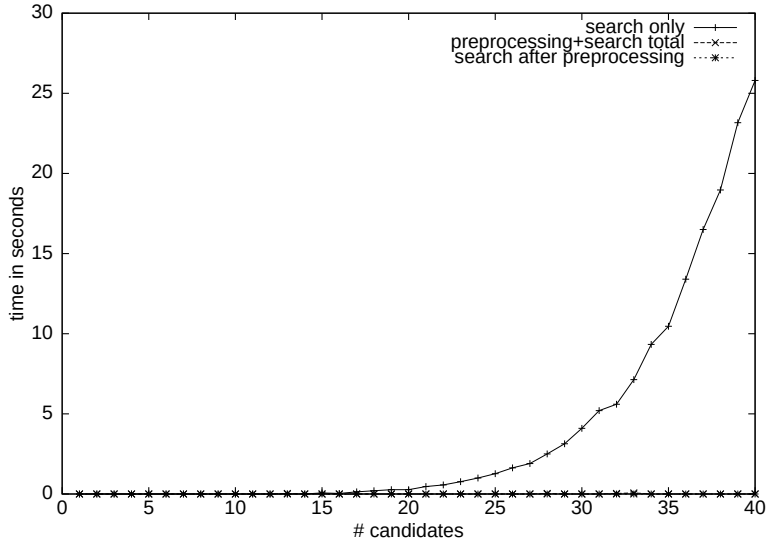


Figure 3.3: 1 issue, 191 voters, 30 instances per data point.

Of course, we do not want to examine only trivial instances. In the next experiment (Figure 3.4), the candidates and voters draw random positions in $[0, 1] \times [0, 1]$; in this two-dimensional setup the voters' preferences are no longer single-peaked.

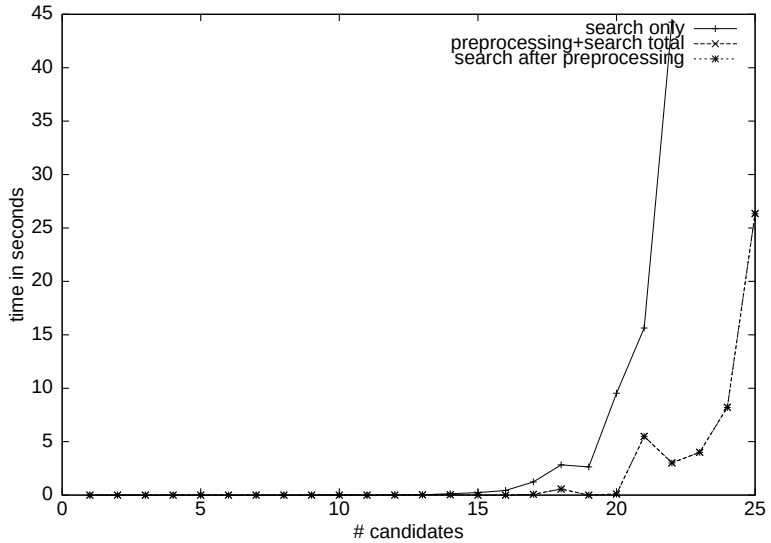


Figure 3.4: 2 issues, 191 voters, 30 instances per data point.

These instances are much harder to solve, but adding the preprocessing technique significantly speeds up the search. We note that essentially no time is spent in the preprocessing stage (the "preprocessing + search total" and "search after preprocessing" curves are essentially identical),

hence the benefits of preprocessing effectively come for free.

We also considered changing the number of votes to a small number. Figure 3.5 shows the results with only 3 votes.

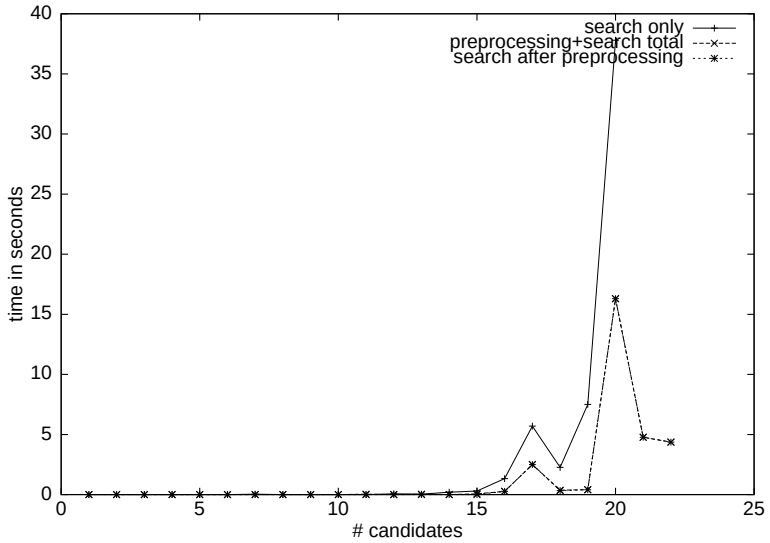


Figure 3.5: 2 issues, 3 voters, 30 instances per data point.

We experimented with introducing additional structure on the set of candidates. In the next experiment, there are 5 *parties* that draw random positions in $[0,1] \times [0,1]$; each candidate randomly chooses a party, and then takes a position that is the average of the party's position and another random point in $[0,1] \times [0,1]$. The results did not change significantly, as shown in Figure 3.6.

We also experimented with having the voters and candidates draw positions on an even larger number of issues (10 issues). Perhaps surprisingly, here the preprocessing technique once again solved all instances immediately (Figure 3.7).

3.1.6 NP-hardness of the Slater problem

In this subsection, we use the technique of sets of similar candidates in an entirely different manner: we show that it is useful in demonstrating the hardness of the Slater problem when there are no pairwise ties. In the case where pairwise ties between candidates are possible, the hardness of the Slater problem follows from the hardness of the Minimum Feedback Edge Set problem. However, as we have already pointed out, most elections do not have pairwise ties (for example, if the number of votes is odd, then there cannot be any pairwise ties). So, how hard is the problem when there are no ties? This problem is equivalent to the Minimum Feedback Edge Set problem on tournament graphs, and was conjectured to be NP-hard as early as 1992 [Bang-Jensen and Thomassen, 1992]. The conjecture remained unproven until 2005, when a randomized reduction was given [Ailon *et al.*, 2005]. A later derandomization of this proof finally proved the conjecture completely [Alon, 2006]. Interestingly, the observations about sets of similar candidates made above allow us to give

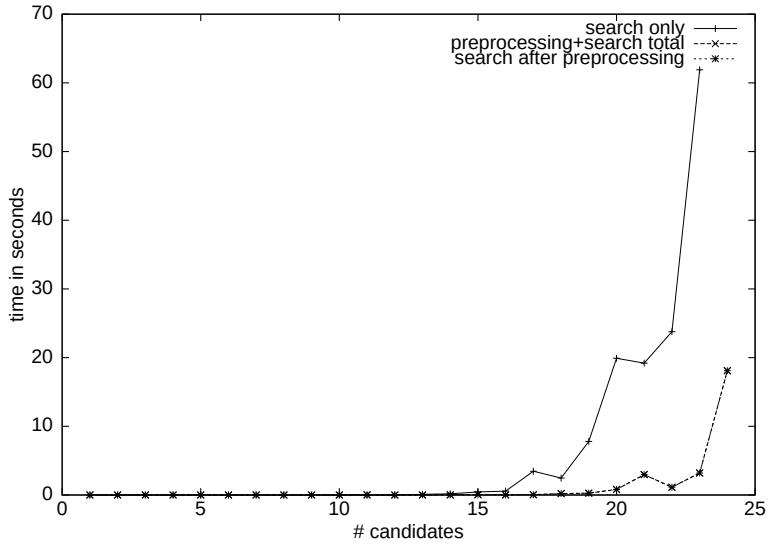


Figure 3.6: 2 issues, 5 parties, 191 voters, 30 instances per data point.

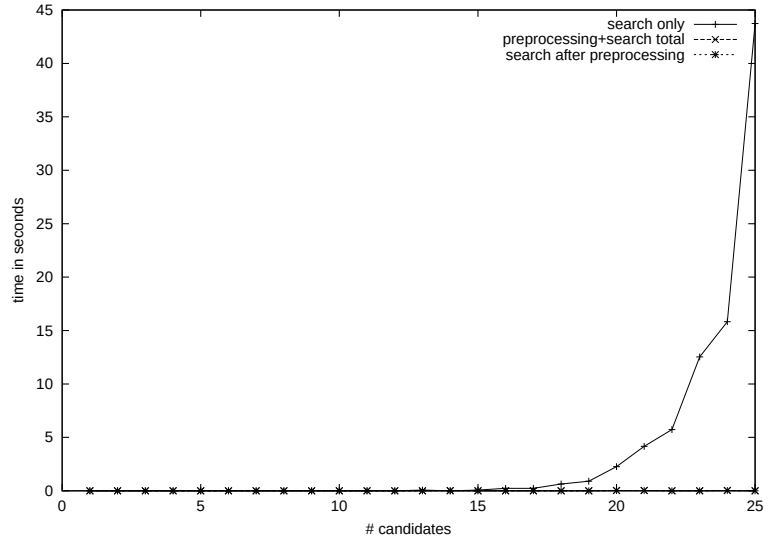


Figure 3.7: 10 issues, 191 voters, 30 instances per data point.

a more direct proof of this result (which does not rely on derandomizing a randomized reduction).[3]

Theorem 3 *The* Slater *problem is NP-hard (even in the absence of pairwise ties).*

[3]Interestingly, the previous reductions [Ailon *et al.*, 2005; Alon, 2006] also use what is effectively an extremely special case of the results about similar candidates presented in this section. That special case is, however, not sufficient for the reduction given here.

Proof: We reduce from the NP-complete Maximum Satisfiability (MAXSAT) problem. We show how to reduce an arbitrary MAXSAT instance, given by a set of clauses K over a set of variables V, and a target number t_1 of clauses to satisfy, to an instance of the Slater problem and a target score t_2, such that there is an ranking with Slater score at least t_2 if and only if there is a solution to the MAXSAT instance (that satisfies at least t_1 clauses). Let M be a sufficiently large number ($M > 6|K||V| + |K|^2$). For every variable $v \in V$, let there be the following 6 super-candidates, each of size M (that is, representing M individual candidates): $C_v = \{a_v, +_v, -_v, b_v, d_v, e_v\}$.[4] Let the individual candidates that any given single super-candidate represents constitute an acyclic pairwise election graph, so that we can order them perfectly and obtain a Slater score of $M(M-1)/2$. Let the super-candidates have the following relationships to each other in the aggregated graph:

- Fix some order $>$ over the variables (e.g. $x_1 > x_2 > \ldots > x_{|V|}$). Then, for any two super-candidates $c_v \in C_v, c_{v'} \in C_{v'}$ with $v > v'$, $c_v \to c_{v'}$.
- For any $v \in V$, for any $c_v \in \{a_v, +_v, -_v\}$ and $c'_v \in \{b_v, d_v, e_v\}$, $c_v \to c'_v$.
- For any $v \in V$, $a_v \to +_v, +_v \to -_v, -_v \to a_v$; $b_v \to d_v, b_v \to e_v, d_v \to e_v$.

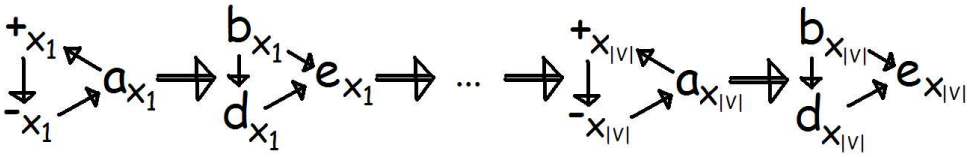


Figure 3.8: Illustration of part of the reduction.

Additionally, for every clause $k \in K$, let there be a single candidate (not a super-candidate) c_k, with the following relationships to the candidates corresponding to variables. Assume without loss of generality that opposite literals ($+v$ and $-v$) never occur in the same clause. Then,

- If $+v \in k$, then $+_v \to c_k, d_v \to c_k, e_v \to c_k$ and $c_k \to a_v, c_k \to -_v, c_k \to b_v$.
- If $-v \in k$, then $-_v \to c_k, d_v \to c_k, e_v \to c_k$ and $c_k \to a_v, c_k \to +_v, c_k \to b_v$.
- If $\{+v, -v\} \cap k = \emptyset$, then $b_v \to c_k, d_v \to c_k, e_v \to c_k$ and $c_k \to a_v, c_k \to +_v, c_k \to -_v$.

The relationships among the c_k are irrelevant. Finally, let the target Slater score be $t_2 = 6|V|M(M-1)/2 + 36M^2|V|(|V|-1)/2 + 14M^2|V| + t_1M$.

We now make some observations about the Slater problem instance that we have constructed. First, by Theorem 1, we can restrict our attention to rankings in which the individual candidates in any given super-candidate form a (contiguous) block. Recall that within such a block, we can order the individual candidates to obtain a Slater score of $M(M-1)/2$, which will give us a total of $6|V|M(M-1)/2$ points. Now, if our ranking of two super-candidates is consistent with the pairwise election graph, according to Lemma 1 this will increase the Slater score by M^2. By contrast, the total number of Slater points that we can obtain from *all* the edges in the pairwise

[4]The letter c is skipped only to avoid confusion with the use of c as an arbitrary candidate.

election graph that involve a candidate c_k corresponding to a clause is at most $|K| \cdot 6|V|M + |K|^2 < 6|K||V|M + |K|^2 M < M^2$. Hence, it is never worth it to sacrifice an agreement on an edge involving two super-candidates to obtain a better result with regard to the remaining candidates, and therefore we can initially restrict our attention to the super-candidates only as these are our primary concern. It is clear that for $v > v'$ we should rank all the candidates in C_v above all those in $C_{v'}$. Doing this for all variables will increase the Slater score by $36M^2|V|(|V|-1)/2$. Moreover, it is clear that for every v we should rank all the candidates in $\{a_v, +_v, -_v\}$ above all those in $\{b_v, d_v, e_v\}$, and $b_v \succ d_v \succ e_v$. Doing this for all variables will increase the Slater score by $(9M^2 + 3M^2)|V| = 12M^2|V|$. Finally, for every v, any one of the rankings $+_v \succ -_v \succ a_v$, $-_v \succ a_v \succ +_v$, and $a_v \succ +_v \succ -_v$ are equally good, leaving us a choice. Choosing one of these for all variables increases the Slater score by another $2M^2|V|$.

Now, as a secondary concern, we can analyze edges involving the c_k. Agreement on an edge between a c_k and one of the super-candidates will increase the Slater score by M. By contrast, the total number of Slater points that we can obtain from *all* the edges in the pairwise election graph that involve only candidates c_k is at most $|K|(|K|-1)/2 < |K|^2 < M$. Hence, it is never worth it to sacrifice an agreement on an edge involving a super-candidate to obtain a better result with regard to the edges involving only candidates c_k, and hence we can restrict our attention to edges involving a super-candidate. (In fact, the edges involving only candidates c_k will turn out to have such a minimal effect on the total score that we need not consider them at all.) Now, we note that whether a candidate c_k is ranked before *all* the candidates in C_v or after *all* of them makes no difference to the total score, because three of these candidates will have an edge into c_k, and three of them will have have an edge out of c_k. Nevertheless, a candidate c_k could be ranked *among* the candidates C_v for (at most) one $v \in V$. Because d_v and e_v always have edges into c_k and are always ranked last among the candidates in C_v, ranking c_k after at least two of the candidates in C_v will never make a positive contribution to the Slater score. Hence, there are only two possibilities to increase the Slater score (by exactly M) for a given c_k: either rank c_k directly after some $+_v$ such that $+v \in k$ and $+_v$ is ranked first among the C_v, or rank c_k directly after some $-_v$ such that $-v \in k$ and $-_v$ is ranked first among the C_v. Of course, for each variable v, we can rank at most one of $+_v$ and $-_v$ first. (We can also rank a_v first, but this will never help us.) Now we can see that this corresponds to the MAXSAT problem: say that we set v to *true* if $+_v$ is ranked first, and to *false* if $-_v$ is ranked first. Then, we can obtain an additional M points for a candidate c_k if and only if clause k is satisfied, and hence we can increase the Slater score by an additional $t_1 M$ points if and only if we can set the variables in such a way as to satisfy at least t_1 clauses. ■

3.1.7 Extension to the Kemeny rule

The *Kemeny* rule [Kemeny, 1959] has significant similarities to the Slater rule. One definition of the Kemeny rule that makes this clear is the following. Instead of minimizing the number of pairwise elections that the final ranking disagrees with, the Kemeny rule tries to minimize the total *weight* of such pairwise elections—where the weight of a pairwise election is the number of votes by which its winner defeated its loser. This also shows that the Kemeny ranking problem is more general than the Slater ranking problem, because it is easy to create votes that make all weights equal to each other. (For example, adding the votes $c_1 \succ c_2 \succ \ldots \succ c_m$ and $c_m \succ c_{m-1} \succ \ldots \succ c_3 \succ c_1 \succ c_2$

gives c_1 two extra votes in its pairwise election against c_2, but has a neutral effect on every other pairwise election.)

The Kemeny rule has an important interpretation as a maximum likelihood estimator of the "correct" ranking. Condorcet, an early social choice theorist, modeled elections as follows: there is a correct ranking of the candidates, but every voter only has a noisy perception of this correct ranking. Specifically, for every pair of candidates, any voter ranks the better candidate higher with probability $p > 1/2$, independently.[5] Given this noise model, the problem of finding the maximum likelihood estimate of the correct outcome, given the votes, is well-defined. Condorcet solved this problem for the cases of 2 and 3 candidates [de Caritat (Marquis de Condorcet), 1785]. Over two centuries later, Young [1995] observed that the Kemeny rule is in fact the solution to Condorcet's problem for arbitrary numbers of candidates. Because of this, the Kemeny rule is sometimes also referred to as the Kemeny-Young rule. We recently showed that some, but not all, of the other well-known voting rules can also be interpreted as maximum likelihood estimators, under different (more complex) noise models [Conitzer and Sandholm, 2005a].

The techniques presented in this section can be extended to apply to the Kemeny rule as well. However, to apply to the Kemeny rule, the definition of a set of similar candidates must be modified to state that for any fixed candidate outside the set, all candidates inside the set must receive exactly the same number of votes in the pairwise election against that candidate (rather than merely obtain the same overall result). This modified definition is much less likely to apply than the original version.

This concludes the part of this dissertation studying the complexity of executing voting rules; we will return to voting, specifically to the hardness of *manipulating* elections, in a few chapters, in Section 8.2. In the next section, we study the complexity of the winner determination problem in combinatorial auctions.

3.2 Combinatorial auctions with structured item graphs

This section describes work that we did jointly with Jonathan Derryberry (CMU).

As we mentioned at the beginning of this chapter, the winner determination problem in a general combinatorial auction (CA) is NP-complete [Rothkopf *et al.*, 1998]. Three main approaches have been pursued to address this: 1) designing optimal search algorithms that are often fast (*e.g.*, [Sandholm, 2002a; Sandholm *et al.*, 2005c; Gonen and Lehmann, 2000; Fujishima *et al.*, 1999; Boutilier, 2002]), but require exponential time in the worst case (unless P = NP), 2) designing approximation algorithms (*e.g.*, [Hoos and Boutilier, 2000; Zurel and Nisan, 2001; van Hoesel and Müller, 2001])—but unfortunately no polytime algorithm can guarantee an approximation (unless ZPP = NP) [Sandholm, 2002a], and 3) designing optimal polytime algorithms for restricted classes of CAs (e.g., [Rothkopf *et al.*, 1998; Tennenholtz, 2000; Penn and Tennenholtz, 2000; Sandholm and Suri, 2003]).

[5] Of course, the rankings of pairs of candidates cannot actually be completely independent, because if a is preferred to b, and b to c, then a must be preferred to c. Nevertheless, all rankings receive some probability under this model, which is all that is necessary for the maximum likelihood approach.

This section falls roughly within the third approach: we present hardness and easiness results for natural classes of CAs. However, we also develop a problem instance parameter (treewidth of the item graph, described below), such that *any* CA instance falls within our framework, and winner determination complexity is exponential in the parameter only. Like almost all of the work on polynomial-time solvable combinatorial auctions, we restrict our attention to the case where any two bids on disjoint subsets can be simultaneously accepted—that is, no XOR-constraints between bids are allowed. (This can be circumvented by adding dummy items that encode these constraints, as discussed in Section 2.2. However, the additional dummy items may significantly increase the time required to solve the instance.)

Consider graphs with the auction's items as vertices, which have the property that for any bid, the items occurring in it constitute a connected set in the graph. (For instance, the fully connected graph (with an edge between every pair of items) always has this property.) Such graphs can have potentially useful structure (for example, the graph may be a tree). For any type of structure, one can ask the following two questions: 1) how hard is the clearing problem when we are given a valid item graph with the desired structure? 2) if the graph is not given beforehand, how hard is it to construct a valid item graph with this structure (if it exists)? We will investigate both questions. 1) was previously solved for the special case where the graph is a tree or a cycle [Sandholm and Suri, 2003]; 2) was previously solved for the special case where the graph is a line (as pointed out by Sandholm and Suri [2003], using a result by Korte and Mohring [1989]) or a cycle (as pointed out by Sandholm and Suri [2003], using a result by Eschen and Spinrad [1993]). In each of these cases, a low-order polynomial algorithm was presented.

One practical use of such polynomially detectable and solvable special cases is to incorporate them into optimal search algorithms [Sandholm and Suri, 2003]. At every node of the search tree, we can detect whether the remaining problem is polynomially solvable, and if it is, we use the polynomial special-purpose algorithm for solving it. Otherwise the search will continue into the subtree.

Also, there are two pure uses for graph structures which make questions 1 and 2 easy. First, the auctioneer can decide on the graph beforehand, and allow only bids on connected sets of items. (In this case, 1 is most important, but 2 may also be useful if the auctioneer wants to make sure that bids on certain bundles are allowed.) Second, the auctioneer can allow bids on any bundle; then, once the bids have been submitted, attempt to construct an item graph that is valid for these bids; and finally, clear the auction using this graph. Clearly, the second approach is only practical if real instances are likely to have item graphs with some structure. To a lesser extent, this is also important for the first approach: if bidders must bid on bundles too different from their desired bundles, economic value will be lost.

Fortunately, real-world instances are likely to have graphs that are not fully connected. For instance, an item may receive only isolated bids that do not involve any other items; or an item may always co-occur with the same other item in bids. As a more detailed example, consider a combinatorial auction for tourist activities in the Bay Area. One item for sale may be a ticket to Alcatraz (in San Francisco). Another may be a ticket to the Children's Discovery Museum (in San Jose). A third item may be a Caltrain ticket to get back and forth between the two cities.[6] Supposing

[6]The reason that this example focuses on the Bay Area is that we published the corresponding paper in the proceedings of the AAAI-04 conference, which was held in San Jose.

(for now) that there is no alternative transportation between the two cities, the only bundle that is unlikely to receive a bid is {Alcatraz, Children's Discovery Museum}, because the bidder will need transportation (no matter which city she is based out of for the duration of her visit). Thus, a valid graph for this auction is the line graph in Figure 3.9 (because its only disconnected set is the one we just ruled out).

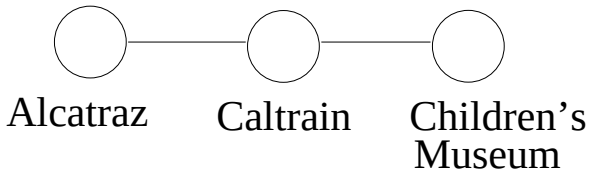


Figure 3.9: A valid item graph.

To extend the example, suppose that there are alternative modes of transportation which are also included in the auction: a Rental Car, and a Bus ticket. Now the following bundles are unlikely to receive a bid: {Alcatraz, Children's Discovery Museum} (because the bidder requires a form of transportation) and any bundle containing more than one mode of transportation. Thus, the graph in Figure 3.10 is a valid item graph (because we just ruled out all its disconnected sets). This graph

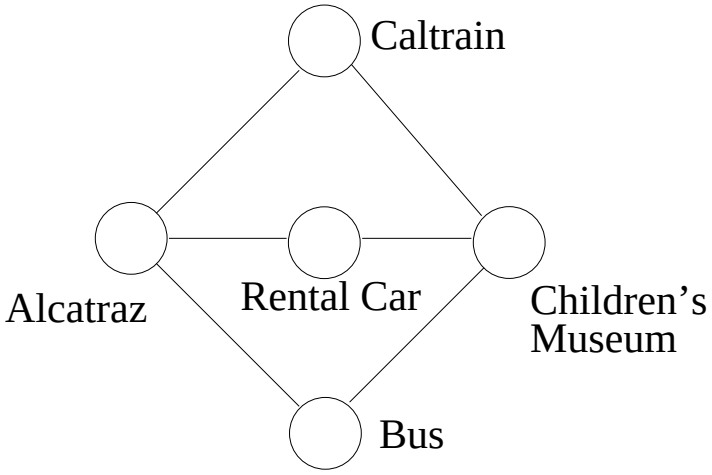


Figure 3.10: Another valid item graph.

does not fall under any of the previously studied structures (it is not a line, tree, or cycle). Still, it has interesting structure: for instance, it has a treewidth of 2.

The rest of this section is organized as follows. We first show that given an item graph with bounded treewidth, the clearing problem can be solved in polynomial time. Next, we show how to

construct an item tree (treewidth = 1), if it exists, in polynomial time. This answers the proposed open question of whether this can be done [Sandholm and Suri, 2003]. (We leave open the question of whether an item graph with small treewidth (say, 2) can be constructed if it exists.) We show that constructing the item graph with the fewest edges is NP-complete (even when a graph of treewidth 2 is easy to construct). Finally, we study a variant where a bid is allowed to consist of k connected sets, rather than just one. We show that the clearing problem is NP-complete even for line graphs with $k = 2$, and the graph construction problem is NP-complete for line graphs with $k = 5$.

3.2.1 Item graphs

We first formally define item graphs.

Definition 11 *Given a combinatorial auction clearing problem instance, the graph $G = (I, E)$, whose vertices correspond to the items in the instance, is a* (valid) item graph *if for every bid, the set of items in that bid constitutes a connected set in G.*

We emphasize that an item graph, in our definition, does *not* need to have an edge connecting every pair of items that occurs in a bid. Rather, each pair only needs to be connected via a path consisting only of items in the bid. In other words, the subgraph consisting of each bid must form only one connected component, but it needs not be a clique.

3.2.2 Clearing with bounded treewidth item graphs

In this subsection, we show that combinatorial auctions can be cleared in polynomial time when an item graph with bounded treewidth is given. This generalizes a result by Sandholm and Suri [Sandholm and Suri, 2003] which shows polynomial time clearability when the item graph is a tree (treewidth = 1).[7] Linear-time approximation algorithms for clearing when the item graph has bounded treewidth have also been given, where the approximation ratio is the treewidth, plus one [Akcoglu *et al.*, 2002]. In contrast, we will clear the auction optimally.

First we will give a very brief review of treewidth.

Definition 12 *A* tree decomposition *T of a graph $G = (I, E)$ is a tree with the following properties.*

1. *Each $v \in T$ has an associated set I_v of vertices in G.*
2. *$\bigcup_{v \in T} I_v = I$ (each vertex of G occurs somewhere in T).*
3. *For each $(i_1, i_2) \in E$, there is some $v \in T$ with $i_1, i_2 \in I_v$ (each edge of G is contained within some vertex of T).*
4. *For each $i \in I$, $\{v \in T : i \in I_v\}$ is connected in T.*

We say that the width *of the tree is* $\max_{v \in T} |I_v| - 1$.

[7] The special case of a tree can also be solved in polynomial time using algorithms for perfect constraint matrices [de Vries and Vohra, 2003], but those algorithms are slower in practice.

While the general problem of finding a tree decomposition of a graph with minimum width is NP-complete, when the treewidth is bounded, the tree decomposition can be constructed in polynomial time [Areborg *et al.*, 1987]. Because we are only interested in the case where the treewidth of the item graph is bounded, we may assume that the tree decomposition is given to us as well as the graph itself.

The following (known) lemma will be useful in our proof.

Lemma 5 *If $X \subseteq I$ is a connected set in G, then $\{v \in T : I_v \cap X \neq \{\}\}$ is connected in T.*

We are now ready to present our first result.

Theorem 4 *Suppose we are given a combinatorial auction problem instance, together with a tree decomposition T with width tw of an item graph G. Then the optimal allocation can be determined in $O(|T|^2(|B|+1)^{tw+1})$ using dynamic programming, where B is the set of bids. (Both with and without free disposal.)*

Proof: Fix a root in T (the "top" of the tree). At every vertex $v \in T$ with items I_v, consider all functions $f : I_v \rightarrow B \cup \{0\}$, indicating possible assignments of the items to the bids. ($f(i) = 0$ indicates no commitment as to which bid item i is assigned to.) This set has size $(|B|+1)^{|I_v|}$. Consider the subset F_{I_v} of these functions satisfying: 1. If $f(i) = b$, b must bid on i; 2. All bids in the image $f(I_v)$ include items that occur higher up in T than v; 3. If $f(i) = b$ and b also bids on item $j \in I_v$, then $f(j) = b$ also.

The interpretation is that each function in F_{I_v} corresponds to a constraint from higher up in the tree as to which bids should be accepted. We now compute, for every node v (starting from the leaves and going up to the root), for every function $f \in F_{I_v}$, the maximum value that can be obtained from items that occur in v and its descendant vertices (but not in any other vertices), and that do not occur in bids in the image of f. (We observe that if v is the root node, there can be no constraints from higher up in the tree (that is, there is only one f function), and the corresponding value is the maximum value that can be obtained in the auction.) Denoting this value by $r(v, f)$, we can compute it using dynamic programming from the leaves up in the following manner:

- Consider all assignments $g : \{i \in I_v : f(i) = 0\} \rightarrow B \cup \{0\}$,[8] with the properties that: 1. If $g(i) = b$, b must bid on i; 2. The image of g does not include any bids that include items that occur higher in T than v. 3. If $g(i) = b$ and b also bids on item $j \in I_v$, then $f(j) = 0$ and $g(j) = b$ also. (Thus, g indicates which bids concerning the unallocated items in I_v we are considering accepting, but only bids that we have not considered higher in the tree.)

- The value of such an assignment is $\sum_{b \in g(I_v)} a(b) + \sum_{w \in T : p(w) = v} r(w, q_w(f, g))$, where $g(I_v)$ is the image of g, $a(b)$ is the value of bid b, $p(w)$ is the parent of w, and $q_w(f, g) : I_w \rightarrow B$ maps items occurring in a bid in the image of either f or g to that bid, and everything else to 0.

- The maximum such value over all g is $r(v, f)$.

[8] In the case of no free disposal, g cannot map to 0.

Because we need to do this computation once for each vertex v in T, the number of assignments g is at most $(|B|+1)^{|I_v|}$ where $|I_v| \leq tw+1$, and for each assignment we need to do a lookup for each of the children of v, this algorithm has running time $O(|T|^2(|B|+1)^{tw+1})$.

The allocation that the algorithm produces can be obtained going back down the tree, as follows: at the root node, there is only one constraint function f mapping everything to 0 (because no bid has items higher up the tree). Thus, consider the r-value maximizing assignment g_{root} for the root; all the bids in its image are accepted. Then, for each of its children, consider the r-value maximizing assignment under the constraint imposed by g_{root}; all the bids in the image of that assignment are also accepted, etc.

To show that the algorithm works correctly, we need to show that no bids that are accepted high up in the tree are "forgotten about" lower in the tree (and its items lower in the tree awarded to other bids). Because the items in a bid constitute a connected set in the item graph G, by Lemma 5, the vertices in T containing items from such a bid are also connected. Now, if a bid b is accepted at the highest vertex $v \in T$ containing an item in b (that is, the items in that vertex occurring in b are awarded to b), each of v's children must also award all its items occurring in b to b; and by the connectedness pointed out above, for each child, either there is at least one such an item in that child, or none of its descendants have any items occurring in b. In the former case, b is also in the image of the child's allocation function, and the same reasoning applies to its children, etc.; in the latter case the fact that b has been accepted is irrelevant to this part of the tree. So, either an accepted bid forces a constraint in a child, and the fact that the bid was accepted is propagated down the tree; or the bid is irrelevant to all that child's descendants as well, and can be safely forgotten about. ■

3.2.3 An algorithm for constructing a valid item tree

So far we discussed question 1: how to clear the auction *given* a valid item graph. In this subsection, we move on to the second question of *constructing* the graph. We present a polynomial-time algorithm that constructs an item tree (that is, an item graph without cycles), if one exists for the bids. This closes a recognized open research problem [Sandholm and Suri, 2003], and is necessary if one wants to use the polynomial item tree clearing algorithm as a subroutine of a search algorithm, as discussed in the introduction.

First, we introduce some notation. In a combinatorial auction with bid set B and item set I, define $items(b) \subseteq I$ to be the set of items in bid b. Also, let T_b refer to the subgraph of a tree containing only vertices represented by $items(b)$ and all edges among elements of $items(b)$.

With these definitions in hand, we are now ready to present the main theorem of this subsection. This theorem shows how to give a tree that "minimally violates" the key requirement of an item graph (namely, that each bid bids on only one component). Thus, if it is actually possible to give a valid item tree, such a tree will be produced by the algorithm.

Theorem 5 *Given an arbitrary set of bids B for items I, a corresponding tree T that minimizes*

$$\sum_{b \in B} \textit{the number of connected components in } T_b$$

can be found in $O(|B| \cdot |I|^2)$ time.

Proof: Consider the algorithm MAKETREE(B, I) shown below, which returns the maximum spanning tree of the complete undirected weighted graph over vertices I in which each edge (i, j) has a weight equal to the number of bids b such that $i, j \in items(b)$.

```
MAKETREE(B, I)
1  A ← An |I| × |I| matrix of 0s
2  for each b in B
3      do for each i in items(b)
4             do for each j ≠ i in items(b)
5                    do A(i, j) ← A(i, j) + 1
6  return the maximum spanning tree of the graph A
```

The running time of MAKETREE(B, I) is $O(|B| \cdot |I|^2)$ from the triply nested **for** loops, plus the time needed to find the maximum spanning tree. The maximum spanning tree can be found in $O(|I|^2)$ time [Cormen *et al.*, 1990], so the running time of the algorithm as a whole is $O(|B| \cdot |I|^2) + O(|I|^2) = O(|B| \cdot |I|^2)$.

To see that MAKETREE(B, I) returns the tree T with the minimum sum of connected components across all T_b, note that the total weight of T can be written as

$$\sum_{b \in B} \text{the number of edges in } T \text{ among } items(b).$$

Because T is a tree, each subgraph T_b is a forest, and the number of edges in any forest equals the number of vertices in the forest, minus the number of components in the forest. It follows that we can rewrite the above expression as

$$s = \sum_{b \in B} |items(b)| - \text{the number of components in } T_b.$$

Because $\sum_{b \in B} |items(b)|$ is a constant, maximizing s is the same as minimizing the sum of the number of connected components across all T_b. ■

In particular, if an item tree exists for the given bids, then in that tree, each bid bids on only one connected component, so the summation in Theorem 5 is equal to the number of bids. Because each term in the summation must always be greater than or equal to 1, this tree minimizes the summation. Thus, MAKETREE will return a tree for which the summation in Theorem 5 is equal to the number of bids as well. But this can only happen if each bid bids on only one connected component. So, MAKETREE will return an item tree.

Corollary 1 MAKETREE *will return a valid item tree if and only if one exists, in* $O(|B| \cdot |I|^2)$ *time. (And whether a tree is a valid item tree can be checked in* $O(|B| \cdot |I|)$ *time.)*

Implications for bid sets without an item tree

The above result presents an algorithm for constructing a tree T from a set of bids that minimizes the sum of the number of connected components across all T_b. Even when the tree returned is not a valid item tree, we can still use it to help us clear the auction, as follows. Suppose MAKETREE

was "close" to being able to construct an item tree, in the sense that only a few bids were split into multiple components. Then, we could use brute force to determine which of these split bids to accept (we could search over all subsets of these split bids), and solve the rest of the problem using dynamic programming as in [Sandholm and Suri, 2003]. If the number of split bids is k, this algorithm takes $O(2^k \cdot |B| \cdot |I|)$ time (so it is efficient if k is small). We note, however, that MAKETREE(B, I) does not minimize the number of split bids (k), as would be desirable for the proposed search technique. Rather, it minimizes the total number of components (summed over bids). Thus, it may prefer splitting many bids into few components each, over splitting few bids into many components each. So there may exist trees that have fewer split bids than the tree returned by MAKETREE (and it would be interesting to try to come up with other algorithms that try to minimize the number of split bids).

Also, MAKETREE does not solve the general problem of constructing an item graph of small treewidth if one exists. The straightforward adaptation of the MAKETREE algorithm to finding an item graph of treewidth 2 (where we find the maximum spanning graph of treewidth 2 in the last step) does not always provide a valid item graph, even when a valid item graph of treewidth 2 exists. To see why, consider an auction instance for which the graph in Figure 3.11 is the unique item graph of treewidth 2 (for example, because for each edge, there is a bid on only its two endpoints). If there

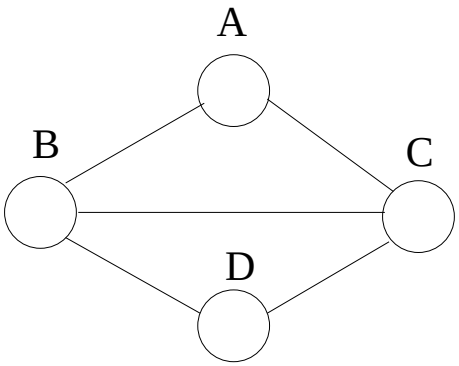


Figure 3.11: A counterexample.

are many bids on the bundle $\{A, B, D\}$, and few other bids, the adapted algorithm will draw the edge (A, D). As a result, it will fail to draw one of the other edges (because otherwise the graph would have treewidth 3), and thus the graph will not be a valid item graph.

For now, we leave open the question of how to construct a valid item graph with treewidth 2 (or 3, or 4, ...) if one exists. However, in the next subsection, we solve a related question.

3.2.4 Constructing the item graph with the fewest edges is hard

The more edges an item graph has, the less structure there is in the instance. A natural question is therefore to construct the valid item graph with the fewest edges. It should be pointed out that this is not necessarily the best graph to work on. For example, given our algorithm, a graph of treewidth 2 may be more desirable to work on than a graph with fewer edges of high treewidth. On

the other hand, assuming that the items cannot be disjoint into two separate components (which is easy to check), a tree is always a graph with the minimum number of edges (and if a tree exists, then only trees have the minimum number of edges). So in this case, generating a graph of minimum treewidth is the same as generating a graph with the minimum number of edges.

We next show that constructing the graph with the fewest edges is hard. Interestingly, the question is hard already for instances with treewidth 2. (For instances of treewidth 1 (forests) it is easy: divide the items into as many separate components (with no bids across more than one component) as possible, and run our MAKETREE algorithm on each.) Thus, if a graph of treewidth 2 can be constructed in polynomial time (and P≠NP), the algorithm for doing so cannot be used to get the fewest edges—unlike the case of treewidth 1.

Theorem 6 *Determining whether an item tree with fewer than q edges exists is NP-complete, even when an item graph of treewidth 2 is guaranteed to exist and each bid is on at most 5 items, and whether or not the item tree we construct is required to be of treewidth 2.*

Proof: The problem is in NP because we can nondeterministically generate a graph with the items as vertices and at most k edges, and check whether it is valid item graph. To show that the problem is NP-complete, we reduce an arbitrary 3SAT instance to the following set of items and bids. For every variable $v \in V$, let there be two items i_{+v}, i_{-v}. Furthermore, let there be two more items, i_0 and i_1. Let the set of bids be as follows. For every $v \in V$, let there be bids on the following sets: $\{i_0, i_{+v}\}, \{i_0, i_{-v}\}, \{i_{+v}, i_{-v}\}, \{i_{+v}, i_{-v}, i_1\}$. Finally, for every clause $c \in C$, let there be a bid on $\{i_{+v} : +v \in c\} \cup \{i_{-v} : -v \in c\} \cup \{i_0, i_1\}$ (the set of all items corresponding to literals in the clause, plus the two extra items—we note that because we are reducing from 3SAT, these are at most 5 items). Let the target number of edges be $q = 4|V|$. We proceed to show that the two instances are equivalent.

First, suppose there exists a solution to the 3SAT instance. Then, let there be an edge between any two items which constitute a bid by themselves; additionally, let there be an edge between i_{+v} and i_1 whenever v is set to *true* in the SAT solution, and an edge between i_{-v} and i_1 whenever v is set to *false* in the SAT solution (for a total of $4|V|$ edges). We observe that all the bids of the form $\{i_{+v}, i_{-v}, i_1\}$ are now connected. Also, for any $c \in C$, because the 3SAT solution satisfied c, either i_1 is connected to some i_{+v} with $+v \in c$, or i_1 is connected to some i_{-v} with $-v \in c$. (And all the items besides i_1 in the bid corresponding to c are clearly connected.) So all the bids constitute connected subsets, and there exists a valid item graph with at most $4|V|$ edges. (Also, this is a series parallel graph, and such graphs have treewidth 2.)

Now, suppose there exists a valid item graph with at most $4|V|$ edges. Of course, there must be an edge between any two items which constitute a bid by themselves; and because of the bids on three items, for every $v \in V$, there must be an edge either between i_{+v} and i_1, or between i_{-v} and i_1. This already requires $4|V|$ edges, so there cannot be any more edges. Because each bid corresponding to a clause c must be connected, there must be either an edge between some i_{+v} with $+v \in c$ and i_1, or between some i_{-v} with $-v \in c$ and i_1. But then, it follows that if we set v to *true* if there is an edge between i_{+v} and i_1, and to *false* if there is an edge between i_{-v} and i_1, we have a solution to the SAT instance.

All that remains to show is that in these instances, a valid item graph of treewidth 2 always exists. Consider the graph that has an edge between any two items which constitute a bid by themselves;

an edge between i_{+v} and i_1 for any $v \in V$; and an edge between i_0 and i_1 (for a total of $4|V| + 1$ edges). This is a series parallel graph, and such graphs have treewidth 2. ■

3.2.5 Applications

The techniques in this section can be applied to any combinatorial auction winner determination instance whose bids happen to be consistent with some (structured) item graph. Specifically, there is no requirement that there is an item graph that makes sense for the items for sale *a priori* (before the bids arrive), based on the inherent properties of the items. Nevertheless, it is important to identify settings in which such *a priori* sensible item graphs do exist, for at least the following reasons. First, as described in the introduction, the auctioneer may wish to guarantee that the techniques described in this section can be applied by disallowing any bids that are not consistent with a prespecified item graph. This prespecified item graph should be chosen to be at least approximately consistent with bidders' likely valuations to minimize the loss of economic value due to this restriction. Second, especially in the case where the number of bidders is large, it is unlikely that the bids will be consistent with any structured item graph, unless the items are fundamentally related to each other in the manner prescribed by such an item graph.

Some settings for which *a priori* sensible item graphs exist have already been proposed. For example, Sandholm and Suri propose a web shopping scenario in which webpages describing the items for sale are structured as a tree, and bidders can, upon deciding that they wish to include the item on a webpage in their bid, continue to browse to neighboring pages [Sandholm and Suri, 2003].

In this subsection, we lay out two new settings in which the inherent relation between the items naturally suggests an item graph with the desired properties. In both of these settings, the "items" for sale do not correspond exactly to the resources under consideration. In the first setting we discuss, combinatorial renting, an item consists of the permission to use a given resource in a given time period. In the second setting, an item consists of a given set of conditions under which a given resource is allocated to the item's winner. We will make this more precise in the following subsubsections.

Combinatorial renting

In a *combinatorial renting auction*, we have a set of resources R and a number of time periods T over which to rent out these resources. In this context, an "item" for sale is a resource-time period pair $(r, t) \in R \times \{1, 2, \ldots, T\}$, and a *bid* consists of a subset of such pairs, representing at which times the bidder wants to rent which resources, together with a value offered for this subset. For example, a company renting out construction equipment may have a cement mixer, a truck, and a crane, each of which can be rented out over the course of three periods. A construction company working on a project may then bid (for example) $(\{(\text{mixer}, 1), (\text{truck}, 2), (\text{crane}, 2), (\text{truck}, 3)\}, \$15000)$, indicating that it wants to rent the mixer in the first period, the truck and the crane in the second period, and the truck again in the third period, for a total value of $15,000.

The constraint that we do not rent the same resource to multiple bidders at the same time corresponds to the constraint that we do not award the same item (r, t) to multiple bidders, and thus the winner determination problem reduces to the standard combinatorial auction winner determi-

nation problem. Moreover, the combinatorial renting auction winner determination problem is in general no easier than the standard combinatorial auction winner determination problem: for example, if $T = 1$, we effectively have a standard combinatorial auction in which items correspond directly to resources. The renting setting becomes more interesting when the multitude of the items is mostly due to the time dimension rather than the resource dimension. Thus, let us assume that the number of resources is small, *i.e.* bounded by a constant k. Additionally, suppose that the bids satisfy the following restriction: the set of time periods in which a bid demands items is connected. That is, for a bid on bundle B, for any $t_1, t_2, t_3 \in \{1, 2, \ldots, T\}, t_1 < t_3 < t_2, r_1, r_2 \in R$ with $(r_1, t_1), (r_2, t_2) \in B$, there must exist some $r_3 \in R$ such that $(r_3, t_3) \in B$. This is a sensible restriction when each bid corresponds to a project for which resources must be rented (*e.g.* the construction example above) and the project is scheduled for a particular time interval.

Under this restriction, the following is a valid item graph:

- For every $t \in \{1, 2, \ldots, T\}$, draw a subgraph G_t whose vertex set is $\{(r, t) : r \in R\}$, and make it fully connected (edges between every pair of vertices).
- Connect the subgraphs by, for every $t \in \{1, 2, \ldots, T-1\}$, drawing an edge from every vertex in G_t to every vertex in G_{t+1}.

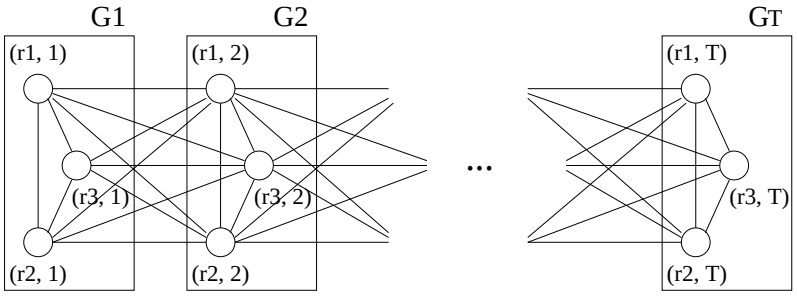


Figure 3.12: Item graph for renting three resources r_1, r_2, r_3.

Moreover, this graph has bounded treewidth as shown by the following tree decomposition (in which V_t is the vertex set of G_t):

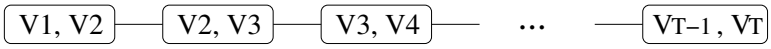


Figure 3.13: Tree decomposition in renting setting.

(In fact, because the tree decomposition is a path, this shows that the original graph has bounded pathwidth.) The width of this decomposition is $2|R| - 1$. This is an *a priori* bound, and it is possible that given the actual bids, an item graph with even smaller treewidth exists.

We now turn to a different setting in which our techniques can be applied.

Conditional awarding of the items

In a *combinatorial conditional awarding auction*, we again have a set of resources R; in addition, we have a set of possible future states of the world S. (For now, we will only concern ourselves with

a single point in the future, say, the beginning of the next fiscal year.) In this context, an "item" for sale is a resource-state pair $(r, s) \in R \times S$, and a *bid* consists of a subset of such pairs, representing under which conditions the bidder wants to be awarded which resources. For example, there may be three states of the world, one in which the price of oil is below \$40 per barrel, one in which the price is between \$40 and \$60, and one in which the price is above \$60. A car dealer may have a sport utility vehicle (SUV) and a small car for sale. Then, a bidder may bid $(\{(\text{SUV}, p_o < \$40), (\text{small car}, p_o < \$40), (\text{SUV}, \$40 \leq p_o \leq \$60), (\text{small car}, p_o > \$60)\}, \$30000)$ indicating that she wants to receive both cars when the price of oil is low, only the SUV when the price of oil is in the middle range, and only the small car when the price of oil is in the high range; and that this total package is worth \$30,000 to her. (In reality, the bidder may wish to make different payments depending on which state of the world materializes; this can be incorporated into the model by using the bidder's *expected* payment.) We note that the "items" for sale are effectively securities that bidders can use to hedge against uncertain future events.

The constraint that we do not award the same resource to multiple bidders in the same state of the world corresponds to the constraint that we do not award the same item (r, s) to multiple bidders, and thus the winner determination problem reduces to the standard combinatorial auction winner determination problem. Also, in the case where $|S| = 1$, we effectively have a standard combinatorial auction in which items correspond directly to resources. Thus, as in the combinatorial renting setting, the most interesting case to look at is where R is small but S is potentially large. One interesting setting to look at is the one in which there is a linear order on the state space S. For example, the "price of oil" state space described above has such an order (higher vs. lower prices). This case is technically similar to the renting problem studied in the previous subsubsection: if the number of items is bounded by a constant k, and the states in which a bid demands items are connected (or there are only small gaps), then we can solve the problem in polynomial time. The connectedness property is likely to hold when the bidder is interested in a set of similar states (for example, the bidder wishes to hedge against high and very high oil prices).

The state space may also be the cross product of multiple linear orders. For example, a state could represent a combination of the price of oil and the exchange rate between the US dollar and the Euro. In this case, we may expect that the set of states in which a bid demands items is connected in a *grid* graph such as the following:

We can turn such a grid into a valid item graph by replacing each state with a fully connected graph (a clique) whose vertices are all items that involve that state (one for every resource), and drawing edges between any pair of items in adjacent cliques. The treewidth of this graph is at most $|R|$ times the treewidth of the grid (the graph over states), because given any tree decomposition of the grid, we can replace each state by all the items involving that state (one for every resource) to obtain a valid tree decomposition of the item graph. Unfortunately, grids do not have bounded treewidth; rather, an $m \times n$ grid has treewidth $\Theta(\min\{m, n\})$. Of course, this is still much better than the trivial bound of mn that would correspond to exhaustive search.

Combining both settings: conditional renting

As a third application, we may wish to combine the applications of the previous two subsubsections and let the bidder *rent* the items *conditionally* on the state of the world at the time of the renting. For example, the construction company described in the first subsubsection may wish to rent different

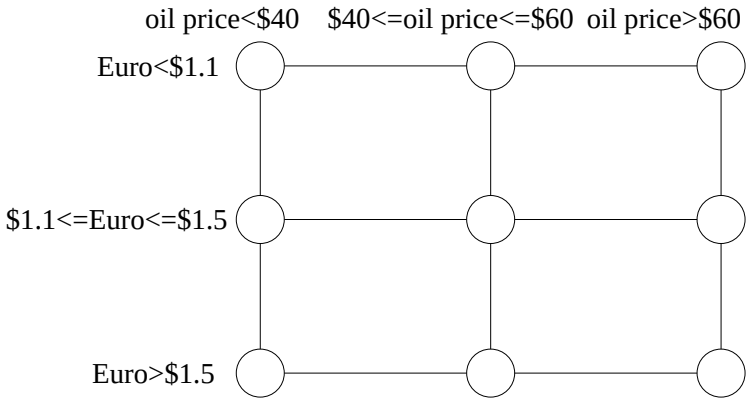


Figure 3.14: State space graph based on oil price and US$/Euro exchange rate.

resources depending on the price of oil (or the weather, or anything else) at the times the resources are to be rented. Hence, the *context* in which a resource is awarded now consists of a (time,state) pair. If there is a linear order on the state space, we may expect that the set of contexts in which a bid demands items is connected in a grid graph such as the following:

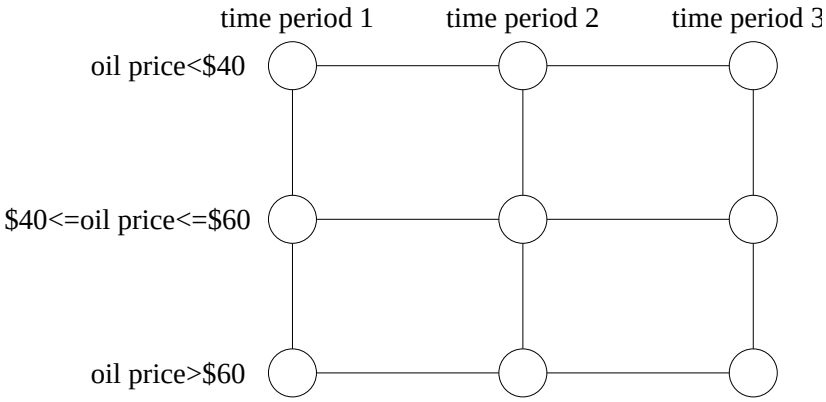


Figure 3.15: Context space graph based on time and state space (oil price).

Technically, this leads to same problem as the two-dimensional state space described in the previous subsubsection.

3.2.6 Bids on multiple connected sets

In this subsection, we investigate what happens if we reduce the requirements on item graphs somewhat. Specifically, let the requirement be that for each bid, the items form *at most* k connected components in the graph. (The case where $k = 1$ is the one we have studied up to this point.) So, to see if a bid is valid given the graph, consider how many connected components the items in the bid constitute in the graph; if (and only if) there are at most k components, the bid is valid. Figure 3.16 shows an example item tree. One bid bids on all the items encapsulated by rectangles (2 connected

components); the other, on all the items encapsulated by ellipses (3 connected components). Thus, if $k = 2$, then the first bid is valid, but the second one is not. (Equivalently, the graph is not valid for the second bid.)

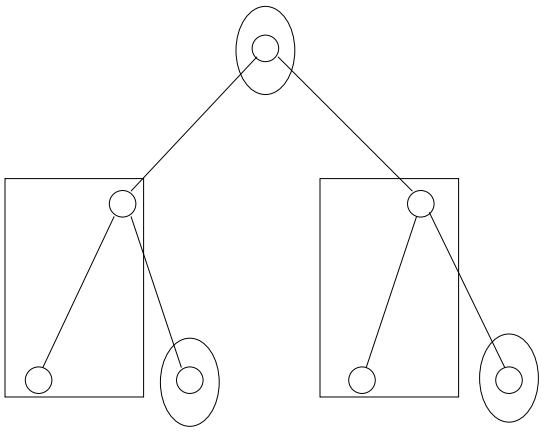


Figure 3.16: An item tree.

As we will see, both clearing when a simple graph is known, and detecting whether a simple graph exists, become hard for $k > 1$.

Clearing is hard even with 2 connected sets on a line graph

Even if the item graph is a line, it is hard to clear auctions in which bidders may bundle two intervals together. To show this, we prove the following slightly stronger theorem.

Theorem 7 *If an item graph is created that consists of two disconnected line graphs, and bidders are permitted to bundle one connected component from each line, then determining if the auction can generate revenue r is NP-complete.*

Proof: The problem is in NP because the general clearing problem is in NP. To show that it is NP-complete, an arbitrary instance of VERTEXCOVER will be reduced to a corresponding auction problem. In VERTEXCOVER, the goal is to determine whether there exists a set of vertices C of size at most k in a graph $G = (V, E)$ such that for each edge $(x, y) \in E$, either $x \in C$ or $y \in C$.

To perform the reduction, given $G = (V, E)$, create an item u_i for each edge e_i. Place all of the u_i items into the upper line. In addition, for each vertex $v_i \in V$, create the items $l_{v_i}^{e_j}$ for each edge e_j that v_i is part of. For each v_i, align all of its corresponding $l_{v_i}^{e_j}$ items into a contiguous interval on the lower line. Now, create the following bids: 1. A bid of price 2 for each "edge item pair" $(l_{v_i}^{e_j}, u_j)$. 2. A bid of price 1 for each "vertex interval bundle," $\{l_{v_i}^{e_j} | \text{for all } e_j \text{ for which } v_i \text{ is part of } e_j\}$.

We now show that there exists a vertex cover of size at most k exactly when the optimal revenue of the corresponding auction is at least $2 \cdot |E| + |V| - k$.

Suppose there is a vertex cover of size k for a graph $G = (V, E)$. Then it is possible to sell all $|E|$ of the edge items breaking up only k of the vertex interval bundles. The resulting revenue if

only those k intervals are rendered unsellable by matching one or more of their members with edge items is $r = 2 \cdot |E| + |V| - k$ as required; the profit is 2 for each of the edge pairs of the form $(l_{v_i}^{e_j}, u_j)$ that are sold, plus 1 for each of the vertex interval bundles that have not had any of their items matched to edges.

Conversely, suppose there is a way to achieve revenue $r = 2 \cdot |E| + |V| - k$. Suppose all of the edge item pairs were sold in this case. Then, because $|V| - k$ vertex intervals were sold, only k were spoiled by selling some of their items with edge pairs. These k vertices can be used as a vertex cover. On the other hand, suppose not all edge pairs were sold. Then the revenue could be increased by selling the rest of the edge pairs even if it means spoiling additional vertex bundles because edge pairs carry a price of 2 while vertex interval bundles only have price 1. This implies that it is possible, by selling all of the edge item pairs, to acheive revenue $r' = 2 \cdot |E| + |V| - k' > r$ so that $k' < k$, where k' represents the size of a possible vertex cover. ■

Corollary 2 *The problem of optimally clearing bids that bundle together at most two connected components of a line item graph is NP-hard because joining the two lines of item graphs of the form discussed in Theorem 7 constitutes a trivially correct reduction.*

Constructing a line graph is hard even with 5 connected sets

We now move on to the task of constructing a simple item graph that is valid when bids are allowed to consist of multiple components. The graph construction question is perhaps less interesting here because, as we just showed, clearing remains hard even if we are given an item line graph. Nevertheless, the graph may still be helpful in reducing the clearing time: maybe the clearing time bound can be reduced to a smaller exponential function, or maybe it can reduce the clearing time in practice. In this subsection, we show that unfortunately, detecting whether a valid line graph exists with multiple (5) components is also NP-complete.

Lemma 6 *Suppose a bid is allowed to contain up to k connected components of items. Then, if there are $m \geq 2k + 5$ items, there exists a set of $O(m^k)$ bids such that there is exactly one line graph (one ordering of the items, up to symmetry) consistent with these bids.*

Proof: Label the items 1 through m. For any subset of $k + 1$ items of which at least two items are successors (i and $i + 1$), let there be a bid on that set of items. We observe that there are at most $(m - 1)\binom{m}{k-1}$ such bids (choosing the successive pair of items first, and then the remaining $k - 1$—of course we are double-counting some combinations this way, but we only want an upper bound), which is $O(m^k)$. Ordering the items $1, 2, \ldots, m$ (or equivalently $m, m - 1, \ldots, 1$), we get a valid item graph (because any two successive items are adjacent in this graph, there are at most k components in every bid). What remains to show is that if the items are ordered differently, there is at least one bid with $k + 1$ components. If the items are ordered differently, there is at least one pair of successive (according to the original labeling) items $i, i + 1$ which are not adjacent in the graph. Consider the set of these two items, plus every item that has an odd index in the ordering of this graph (besides the ones that coincide with, or are adjacent to, the first two). This set has at least $2 + (k + 3) - 4 = k + 1$ items, two of which are adjacent in the original labeling, and each of which

is a separate component. It follows that there exists a subset of this set which constituted one of the bids, and now has $k+1$ components. ■

Lemma 7 *Suppose each bid is allowed to contain at most k connected components, and we have a set of bids that forces a unique ordering of the items (up to symmetry). Then suppose we replace one item r with two new items n_1 and n_2, and let every bid bidding on the original item bid on both the new items. Then the only valid orderings of the new set of items are the valid orderings for the original set, where r is replaced by n_1, n_2 (where these two can be placed in any order). This extends to replacing multiple items by pairs.*

Proof: First we show that these orderings are indeed valid. Clearly, no bid that did not include r will now have more components. Also, no bid that did include r will now have more components, because the component including r is still intact as a single component (since the bid bids on both n_1 and n_2). So the new orderings are valid. Now we will show that these are the only valid orderings. We observe that if we remove one of n_1, n_2 from a given valid ordering as well as from the bids, then we must still have a valid ordering. But because now r has been replaced by a single item, we know that the valid ordering for this is unique (up to symmetry). It follows that n_1 and n_2 had taken r's place in the unique valid ordering. The argument extends straightforwardly to replacing multiple items by item pairs. ■

Theorem 8 *Given the bids, detecting whether an ordering of the items (a line graph) exists such that each bidder bids on at most 5 connected components is NP-complete.*

Proof: The problem is in NP because we can nondeterministically generate an ordering of the items, and check whether any bid is bidding on more than 5 components. To show that the problem is NP-hard, we reduce an arbitrary 3SAT instance to the following sets of items and bids. For every variable $v \in V$, let there be four items, $i_v^*, i_v, i_{+v}, i_{-v}$. Let the set of bids be as follows. First, using Lemma 6 and Lemma 7, let there be $O(m^5)$ bids such that the only remaining valid orderings are $i_{v_1}^*, i_{v_1}, \{i_{+v_1}, i_{-v_1}\}, i_{v_2}^*, i_{v_2}, \{i_{+v_2}, i_{-v_2}\}, \ldots, i_{v_n}^*, i_{v_n}, \{i_{+v_n}, i_{-v_n}\}$. (Here, two items are in set notation if their relative order is not yet determined.) Finally, for every clause $c \in C$, let there be a bid bidding on any i_v with v occurring in c (whether it is $+v$ or $-v$), and on any i_{+v} with $+v$ occurring in c, and on any i_{-v} with $-v$ occurring in c. (So, 6 items in total.) We show the instances are equivalent.

First suppose there exists a solution to the 3SAT instance. Then, whenever a variable v is set to *true*, let i_{+v} be ordered to the left of i_{-v}; otherwise, let i_{+v} be ordered to the right of i_{-v}. Then, for every clause, for some literal $+v$ (or $-v$) occurring in that clause, i_{+v} (or i_{-v}) is adjacent to i_v, and it follows that the bid corresponding to the clause has at most 5 connected components. So, there is a valid ordering.

Now suppose there exists a valid ordering. Because of the i_v^* items, the only items in a bid corresponding to a clause that can possibly be adjacent are an i_{+v} and the corresponding i_v, or an i_{-v} and the corresponding i_v. This must happen at least once for every bid corresponding to a clause (or the bid would have 6 components. But then, if we set a variable v to *true* if i_{+v} and i_v

are adjacent, and to *false* otherwise, every clause must have at least one $+v$ in it where v is set to *true*, or at least one $-v$ in it where v is set to *false*. It follows that there is a solution to the 3SAT instance. ■

This concludes the part of this dissertation studying the complexity of the winner determination problem in combinatorial auctions; we will return to combinatorial auctions and exchanges (specifically, to the use of the VCG mechanism in such settings) in the chapter after the next chapter, Section 5.1. In the next section, we study the complexity of the outcome optimization problem in the setting of expressive preference aggregation for donations to charities.

3.3 Expressive preference aggregation for donations to charities

In this section, we study the outcome optimization problem for expressive preference aggregation for donations to charities, as defined in Section 2.3. We will refer to this problem as the *clearing* problem. The formal definition follows.

Definition 13 (DONATION-CLEARING) *We are given a set of n bids (each given by a utility function for each charity, and a payment willingness function) over charities $c_1, c_2, \ldots, c_m$ as described in Section 2.3. Additionally, we are given an objective function (*e.g. *surplus, or total amount donated). We are asked to find an objective-maximizing valid outcome.*

One aspect of the problem is not captured by this definition: if we want a decentralized solution, in which bidders donate their money to the charity directly (rather than to a center who then redistributes it), then we also need to specify which bidder donates how much to which charity. Assuming that we are given the centralized solution, any greedy algorithm that increases the cash flow from any bidder who has not yet paid enough, to any charity that has not yet received enough, until either the bidder has paid enough or the charity has received enough, will provide such a specification. Recall, however, that we may wish to allow for bidders to state that they do not wish to donate to certain charities. In general, checking whether a given centralized solution can be accomplished through decentralized payments when there are such constraints can be modeled as a MAX-FLOW problem. In the MAX-FLOW instance, there is an edge from the source node s to each bidder b_j, with a capacity of π_{b_j} (as specified in the centralized solution); an edge from each bidder b_j to each charity c_i *that the bidder is willing to donate money to*, with a capacity of ∞; and an edge from each charity c_i to the target node t with capacity π_{c_i} (as specified in the centralized solution).

In the remainder of this section, we will no longer consider the problem of decentralizing solutions; rather, we focus on the DONATION-CLEARING problem. How difficult the DONATION-CLEARING problem is depends on the types of bids used and the language in which they are expressed. This is the topic of the next subsection.

3.3.1 Hardness of clearing the market

In this subsection, we will show that the clearing problem is completely inapproximable, even when every bidder's utility function is linear (with slope 0 or 1 in each charity's payments), each

bidder cares either about at most two charities or about all charities equally, and each bidder's payment willingness function is a step function. We will reduce from MAX2SAT (given a formula in conjunctive normal form (where each clause has two literals) and a target number of satisfied clauses T, does there exist an assignment of truth values to the variables that makes at least T clauses true?), which is NP-complete [Garey *et al.*, 1976].

Theorem 9 *There exists a reduction from MAX2SAT instances to DONATION-CLEARING instances such that 1. If the MAX2SAT instance has no solution, then the only valid outcome is the zero outcome (no bidder pays anything and no charity receives anything); 2. Otherwise, there exists a solution with positive surplus. Additionally, the DONATION-CLEARING instances that we reduce to have the following properties: 1. Every u_j^i is a line; that is, the utility that each bidder derives from any charity is linear; 2. All the u_j^i have slope either 0 or 1; 3. Every bidder either has at most 2 charities that affect her utility (with slope 1), or all charities affect her utility (with slope 1); 4. Every bid is a threshold bid; that is, every bidder's payment willingness function w_j is a step function.*

Proof: The problem is in NP because we can nondeterministically choose the payments to be made and received, and check the validity and objective value of this outcome.

In the following, we will represent bids as follows: $(\{(c_k, a_k)\}, s, t)$ indicates that $u_j^k(\pi_{c_k}) = a_k \pi_{c_k}$ (this function is 0 for c_k not mentioned in the bid), and $w_j(u_j) = t$ for $u_j \geq s$, $w_j(u_j) = 0$ otherwise.

To show NP-hardness, we reduce an arbitrary MAX2SAT instance, given by a set of clauses $K = \{k\} = \{(l_k^1, l_k^2)\}$ over a variable set V together with a target number of satisfied clauses T, to the following DONATION-CLEARING instance. Let the set of charities be as follows. For every literal $l \in L$, there is a charity c_l. Then, let the set of bids be as follows. For every variable v, there is a bid $b_v = (\{(c_{+v}, 1), (c_{-v}, 1)\}, 2, 1 - \frac{1}{4|V|})$. For every literal l, there is a bid $b_l = (\{(c_l, 1)\}, 2, 1)$. For every clause $k = \{l_k^1, l_k^2\} \in K$, there is a bid $b_k = (\{(c_{l_k^1}, 1), (c_{l_k^2}, 1)\}, 2, \frac{1}{8|V||K|})$. Finally, there is a single bid that values all charities equally: $b_0 = (\{(c_1, 1), (c_2, 1), \ldots, (c_m, 1)\}, 2|V| + \frac{T}{8|V||K|}, \frac{1}{4} + \frac{1}{16|V||K|})$. We show the two instances are equivalent.

First, suppose there exists a solution to the MAX2SAT instance. If in this solution, l is *true*, then let $\pi_{c_l} = 2 + \frac{T}{8|V|^2|K|}$; otherwise $\pi_{c_l} = 0$. Also, the only bids that are *not* accepted (meaning the threshold is not met) are the b_l where l is *false*, and the b_k such that both of l_k^1, l_k^2 are *false*. First we show that no bidder whose bid is accepted pays more than she is willing to. For each b_v, either c_{+v} or c_{-v} receives at least 2, so this bidder's threshold has been met. For each b_l, either l is *false* and the bid is not accepted, or l is *true*, c_l receives at least 2, and the threshold has been met. For each b_k, either both of l_k^1, l_k^2 are *false* and the bid is not accepted, or at least one of them (say l_k^i) is *true* (that is, k is satisfied) and $c_{l_k^i}$ receives at least 2, and the threshold has been met. Finally, because the total amount received by the charities is $2|V| + \frac{T}{8|V||K|}$, b_0's threshold has also been met. The total amount that can be extracted from the accepted bids is at least $|V|(1 - \frac{1}{4|V|}) + |V| + T\frac{1}{8|V||K|} + \frac{1}{4} + \frac{1}{16|V||K|}) = 2|V| + \frac{T}{8|V|K|} + \frac{1}{16|V||K|} > 2|V| + \frac{T}{8|V||K|}$, so there is positive surplus. So there exists a solution with positive surplus to the DONATION-CLEARING instance.

Now suppose there exists a nonzero outcome in the DONATION-CLEARING instance. First we show that it is not possible (for any $v \in V$) that both b_{+v} and b_{-v} are accepted. For, this would require that $\pi_{c_{+v}} + \pi_{c_{-v}} \geq 4$. The bids b_v, b_{+v}, b_{-v} cannot contribute more than 3, so we need another 1 at least. It is easily seen that for any other v', accepting any subset of $\{b_{v'}, b_{+v'}, b_{-v'}\}$ would require that at least as much is given to $c_{+v'}$ and $c_{-v'}$ as can be extracted from these bids, so this cannot help. Finally, all the other bids combined can contribute at most $|K|\frac{1}{8|V||K|} + \frac{1}{4} + \frac{1}{16|V||K|} < 1$. It follows that we can interpret the outcome in the DONATION-CLEARING instance as a partial assignment of truth values to variables: v is set to *true* if b_{+v} is accepted, and to *false* if b_{-v} is accepted. All that is left to show is that this partial assignment satisfies at least T clauses.

First we show that if a clause bid b_k is accepted, then either $b_{l_k^1}$ or $b_{l_k^2}$ is accepted (and thus either l_k^1 or l_k^2 is set to *true*, hence k is satisfied). If b_k is accepted, at least one of $c_{l_k^1}$ and $c_{l_k^2}$ must be receiving at least 1; without loss of generality, say it is $c_{l_k^1}$, and say l_k^1 corresponds to variable v_k^1 (that is, it is $+v_k^1$ or $-v_k^1$). If $c_{l_k^1}$ does not receive at least 2, $b_{l_k^1}$ is not accepted, and it is easy to check that the bids $b_{v_k^1}, b_{+v_k^1}, b_{-v_k^1}$ contribute (at least) 1 less than is paid to $c_{+v_k^1}$ and $c_{+v_k^1}$. But this is the same situation that we analyzed before, and we know it is impossible. All that remains to show is that at least T clause bids are accepted.

We now show that b_0 is accepted. Suppose it is not; then one of the b_v must be accepted. (The solution is nonzero by assumption; if only some b_k are accepted, the total payment from these bids is at most $|K|\frac{1}{8|V||K|} < 1$, which is not enough for any bid to be accepted; and if one of the b_l is accepted, then the threshold for the corresponding b_v is also reached.) For this v, $b_{v_k^1}, b_{+v_k^1}, b_{-v_k^1}$ contribute (at least) $\frac{1}{4|V|}$ less than the total payments to c_{+v} and c_{-v}. Again, the other b_v and b_l cannot (by themselves) help to close this gap; and the b_k can contribute at most $|K|\frac{1}{8|V||K|} < \frac{1}{4|V|}$. It follows that b_0 is accepted.

Now, in order for b_0 to be accepted, a total of $2|V| + \frac{T}{8|V||K|}$ must be donated. Because is not possible (for any $v \in V$) that both b_{+v} and b_{-v} are accepted, it follows that the total payment by the b_v and the b_l can be at most $2|V| - \frac{1}{4}$. Adding b_0's payment of $\frac{1}{4} + \frac{1}{16|V||K|}$ to this, we still need $\frac{T-\frac{1}{2}}{8|V||K|}$ from the b_k. But each one of them contributes at most $\frac{1}{8|V||K|}$, so at least T of them must be accepted. ■

Corollary 3 *Unless P=NP, there is no polynomial-time algorithm for approximating DONATION-CLEARING (with either the surplus or the total amount donated as the objective) within any ratio $f(n)$, where f is a nonzero function of the size of the instance. This holds even if the DONATION-CLEARING structures satisfy all the properties given in Theorem 9.*

Proof: Suppose we had such a polynomial time algorithm, and applied it to the DONATION-CLEARING instances that were reduced from MAX2SAT instances in Theorem 9. It would return a nonzero solution when the MAX2SAT instance has a solution, and a zero solution otherwise. So we can decide whether arbitrary MAX2SAT instances are satisfiable this way, and it would follow that P=NP. ■

This should not be interpreted to mean that our approach is infeasible. First, as we will show, there are very expressive families of bids for which the problem is solvable in polynomial time.

Second, NP-completeness is often overcome in practice (especially when the stakes are high). For instance, even though the problem of clearing combinatorial auctions is NP-complete [Rothkopf *et al.*, 1998] (even to approximate [Sandholm, 2002a]), they are typically solved to optimality in practice [Sandholm *et al.*, 2006; Sandholm, 2006].

3.3.2 Mixed integer programming formulation

In this subsection, we give a mixed integer programming (MIP) formulation for the general problem. We also discuss in which special cases this formulation reduces to a linear programming (LP) formulation. In such cases, the problem is solvable in polynomial time, because linear programs can be solved in polynomial time [Khachiyan, 1979].

The variables of the MIP defining the final outcome are the payments made to the charities, denoted by π_{c_i}, and the payments extracted from the bidders, π_{b_j}. In the case where we try to avoid direct payments and let the bidders pay the charities directly, we add variables π_{c_i,b_j} indicating how much b_j pays to c_i, with the constraints that for each c_i, $\pi_{c_i} \leq \sum_{b_j} \pi_{c_i,b_j}$; and for each b_j, $\pi_{b_j} \geq \sum_{c_i} \pi_{c_i,b_j}$. Additionally, there is a constraint $\pi_{c_i,b_j} = 0$ whenever bidder b_j is unwilling to pay charity c_i. The rest of the MIP can be phrased in terms of the π_{c_i} and π_{b_j}.

The objectives we have discussed earlier are both linear: surplus is given by $\sum_{j=1}^{n} \pi_{b_j} - \sum_{i=1}^{m} \pi_{c_i}$, and total amount donated is given by $\sum_{i=1}^{m} \pi_{c_i}$ (coefficients can be added to represent different weights on the different charities in the objective).

The constraint that the outcome should be valid (no deficit) is given simply by: $\sum_{j=1}^{n} \pi_{b_j} \geq \sum_{i=1}^{m} \pi_{c_i}$.

For every bidder, for every charity, we define an additional utility variable u_j^i indicating the utility that this bidder derives from the payment to this charity. The bidder's total utility is given by another variable u_j, with the constraint that $u_j = \sum_{i=1}^{m} u_j^i$.

Each u_j^i is given as a function of π_{c_i} by the (piecewise linear) function provided by the bidder. In order to represent this function in the MIP formulation, we will merely place upper bounding constraints on u_j^i, so that it cannot exceed the given functions. The MIP solver can then push the u_j^i variables all the way up to the constraint, in order to extract as much payment from this bidder as possible. In the case where the u_j^i are concave, this is easy: if (s_l, t_l) and (s_{l+1}, t_{l+1}) are endpoints of a finite linear segment in the function, we add the constraint that $u_j^i \leq t_l + \frac{\pi_{c_i} - s_l}{s_{l+1} - s_l}(t_{l+1} - t_l)$. If the final (infinite) segment starts at (s_k, t_k) and has slope d, we add the constraint that $u_j^i \leq t_k + d(\pi_{c_i} - s_k)$. Using the fact that the function is concave, for each value of π_{c_i}, the tightest upper bound on u_j^i is the one corresponding to the segment above that value of π_{c_i}, and therefore these constraints are sufficient to force the correct value of u_j^i.

When the function is not concave, we require (for the first time) some binary variables. First, we define another point on the function: $(s_{k+1}, t_{k+1}) = (s_k + M, t_k + dM)$, where d is the slope of

the infinite segment and M is any upper bound on the π_{c_j}. This has the effect that we will never be on the infinite segment again. Now, let $x_l^{i,j}$ be an indicator variable that should be 1 if π_{c_i} is below the lth segment of the function, and 0 otherwise. To effect this, first add a constraint $\sum_{l=0}^{k} x_l^{i,j} = 1$. Now, we aim to represent π_{c_i} as a weighted average of its two neighboring $s_l^{i,j}$. For $0 \leq l \leq k+1$, let $\lambda_l^{i,j}$ be the weight on $s_l^{i,j}$. We add the constraint $\sum_{l=0}^{k+1} \lambda_l^{i,j} = 1$. Also, for $0 \leq l \leq k+1$, we add the constraint $\lambda_l^{i,j} \leq x_{l-1} + x_l$ (where x_{-1} and x_{k+1} are defined to be zero), so that indeed only the two neighboring $s_l^{i,j}$ have nonzero weight. Now we add the constraint $\pi_{c_i} = \sum_{l=0}^{k+1} s_l^{i,j} \lambda_l^{i,j}$, and now the $\lambda_l^{i,j}$ must be set correctly. Then, we can set $u_j^i = \sum_{l=0}^{k+1} t_l^{i,j} \lambda_l^{i,j}$. (This is a standard MIP technique [Nemhauser and Wolsey, 1999].)

Finally, each π_{b_j} is bounded by a function of u_j by the (piecewise linear) function provided by the bidder (w_j). Representing this function is entirely analogous to how we represented u_j^i as a function of π_{c_i}. (Again we will need binary variables only if the function is not concave.)

Because we only use binary variables when either a utility function u_j^i or a payment willingness function w_j is not concave, it follows that if all of these are concave, our MIP formulation is simply a linear program—which can be solved in polynomial time. Thus:

Theorem 10 *If all functions u_j^i and w_j are concave (and piecewise linear), the DONATION-CLEARING problem can be solved in polynomial time using linear programming.*

Even if some of these functions are not concave, we can simply replace each such function by the smallest upper bounding concave function, and use the linear programming formulation to obtain an upper bound on the objective—which may be useful in a search formulation of the general problem.

3.3.3 Why one cannot do much better than linear programming

One may wonder if, for the special cases of the DONATION-CLEARING problem that can be solved in polynomial time with linear programming, there exist special-purpose algorithms that are much faster than linear programming algorithms. In this subsection, we show that this is not the case. We give a reduction *from* (the decision variant of) the general linear programming problem to (the decision variant of) a special case of the DONATION-CLEARING problem (which can be solved in polynomial time using linear programming). (The decision variant of an optimization problem asks the binary question: "Can the objective value exceed o?") Thus, any special-purpose algorithm for solving the decision variant of this special case of the DONATION-CLEARING problem could be used to solve a decision question about an arbitrary linear program just as fast. (And thus, if we are willing to call the algorithm a logarithmic number of times, we can solve the optimization version of the linear program.)

We first observe that for linear programming, a decision question about the objective can simply be phrased as another constraint in the LP (forcing the objective to exceed the given value); then, the

original decision question coincides with asking whether the resulting linear program has a feasible solution.

Theorem 11 *The question of whether an LP (given by a set of linear constraints[9]) has a feasible solution can be modeled as a DONATION-CLEARING instance with payment maximization as the objective, with $2v$ charities and $v + c$ bids (where v is the number of variables in the LP, and c is the number of constraints). In this model, each bid b_j has only linear u_j^i functions, and is a partially acceptable threshold bid ($w_j(u) = t_j$ for $u \geq s_j$, otherwise $w_j(u) = \frac{ut_j}{s_j}$). The v bids corresponding to the variables mention only two charities each; the c bids corresponding to the constraints mention only two times the number of variables in the corresponding constraint.*

Proof: For every variable x_i in the LP, let there be two charities, c_{+x_i} and c_{-x_i}. Let H be some number such that if there is a feasible solution to the LP, there is one in which every variable has absolute value at most H.

In the following, we will represent bids as follows: $(\{(c_k, a_k)\}, s, t)$ indicates that $u_j^k(\pi_{c_k}) = a_k\pi_{c_k}$ (this function is 0 for c_k not mentioned in the bid), and $w_j(u_j) = t$ for $u_j \geq s$, $w_j(u_j) = \frac{u_j t}{s}$ otherwise.

For every variable x_i in the LP, let there be a bid $b_{x_i} = (\{(c_{+x_i}, 1), (c_{-x_i}, 1)\}, 2H, 2H - \frac{c}{v})$. For every constraint $\sum_i r_i^j x_i \leq s_j$ in the linear program, let there be a bid $b_j = (\{(c_{-x_i}, r_i^j)\}_{i:r_i^j>0} \cup \{(c_{+x_i}, -r_i^j)\}_{i:r_i^j<0}, (\sum_i |r_i^j|)H - s_j, 1)$. Let the target total amount donated be $2vH$.

Suppose there is a feasible solution $(x_1^*, x_2^*, \ldots, x_v^*)$ to the LP. Without loss of generality, we can suppose that $|x_i^*| \leq H$ for all i. Then, in the DONATION-CLEARING instance, for every i, let $\pi_{c_{+x_i}} = H + x_i^*$, and let $\pi_{c_{-x_i}} = H - x_i^*$ (for a total payment of $2H$ to these two charities). This allows us to extract the maximum payment from the bids b_{x_i}—a total payment of $2vH - c$. Additionally, the utility of bidder b_j is now $\sum_{i:r_i^j>0} r_i^j(H - x_i^*) + \sum_{i:r_i^j<0} -r_i^j(H + x_i^*) = (\sum_i |r_i^j|)H - \sum_i r_i^j x_i^* \geq (\sum_i |r_i^j|)H - s_j$ (where the last inequality stems from the fact that constraint j must be satisfied in the LP solution), so it follows we can extract the maximum payment from all the bidders b_j, for a total payment of c. It follows that we can extract the required $2vH$ payment from the bidders, and there exists a solution to the DONATION-CLEARING instance with a total amount donated of at least $2vH$.

Now suppose there is a solution to the DONATION-CLEARING instance with a total amount donated of at least vH. Then the maximum payment must be extracted from each bidder. From the fact that the maximum payment must be extracted from each bidder b_{x_i}, it follows that for each i, $\pi_{c_{+x_i}} + \pi_{c_{-x_i}} \geq 2H$. Because the maximum extractable total payment is $2vH$, it follows that for each i, $\pi_{c_{+x_i}} + \pi_{c_{-x_i}} = 2H$. Let $x_i^* = \pi_{c_{+x_i}} - H = H - \pi_{c_{-x_i}}$. Then, from the fact that the maximum payment must be extracted from each bidder b_j, it follows that $(\sum_i |r_i^j|)H - s_j \leq \sum_{i:r_i^j>0} r_i^j \pi_{c_{-x_i}} + \sum_{i:r_i^j<0} -r_i^j \pi_{c_{+x_i}} = \sum_{i:r_i^j>0} r_i^j(H - x_i^*) + \sum_{i:r_i^j<0} -r_i^j(H + x_i^*) = (\sum_i |r_i^j|)H - \sum_i r_i^j x_i^*$.

[9]These constraints must include bounds on the variables (including nonnegativity bounds), if any.

Equivalently, $\sum_i r_i^j x_i^* \leq s_j$. It follows that the x_i^* constitute a feasible solution to the LP. ■

3.3.4 Quasilinear bids

Another class of bids of interest is the class of *quasilinear bids*. In a quasilinear bid, the bidder's payment willingness function is linear in utility: that is, $w_j = u_j$. (Because the units of utility are arbitrary, we may as well let them correspond exactly to units of money—so we do not need a constant multiplier.) In most cases, quasilinearity is an unreasonable assumption: for example, usually bidders have a limited budget for donations, so that the payment willingness will stop increasing in utility after some point (or at least increase slower in the case of a "softer" budget constraint). Nevertheless, quasilinearity may be a reasonable assumption in the case where the bidders are large organizations with large budgets, and the charities are a few small projects requiring relatively little money. In this setting, once a certain small amount has been donated to a charity, a bidder will derive no more utility from more money being donated from that charity. Thus, the bidders will never reach a high enough utility for their budget constraint (even when it is soft) to take effect, and thus a linear approximation of their payment willingness function is reasonable. Another reason for studying the quasilinear setting is that it is the easiest setting for mechanism design, which we will discuss shortly. In this subsection, we will see that the clearing problem is much easier in the case of quasilinear bids.

First, we address the case where we are trying to maximize surplus (which is the most natural setting for mechanism design). The key observation here is that when bids are quasilinear, the clearing problem *decomposes* across charities.

Lemma 8 *Suppose all bids are quasilinear, and surplus is the objective. Then we can clear the market optimally by clearing the market for each charity individually. That is, for each bidder b_j, let $\pi_{b_j} = \sum_{c_i} \pi_{b_j^i}$. Then, for each charity c_i, maximize $(\sum_{b_j} \pi_{b_j^i}) - \pi_{c_i}$, under the constraint that for every bidder b_j, $\pi_{b_j^i} \leq u_j^i(\pi_{c_i})$.*

Proof: The resulting solution is certainly valid: first of all, at least as much money is collected as is given away, because $\sum_{b_j} \pi_{b_j} - \sum_{c_i} \pi_{c_i} = \sum_{b_j} \sum_{c_i} \pi_{b_j^i} - \sum_{c_i} \pi_{c_i} = \sum_{c_i}((\sum_{b_j} \pi_{b_j^i}) - \pi_{c_i})$—and the terms of this summation are the objectives of the individual optimization problems, each of which can be set at least to 0 (by setting all the variables are set to 0), so it follows that the expression is nonnegative. Second, no bidder b_j pays more than she is willing to, because $u_j - \pi_{b_j} = \sum_{c_i} u_j^i(\pi_{c_i}) - \sum_{c_i} \pi_{b_j^i} = \sum_{c_i}(u_j^i(\pi_{c_i}) - \pi_{b_j^i})$—and the terms of this summation are nonnegative by the constraints we imposed on the individual optimization problems.

All that remains to show is that the solution is optimal. Because in an optimal solution, we will extract as much payment from the bidders as possible given the π_{c_i}, all we need to show is that the π_{c_i} are set optimally by this approach. Let $\pi_{c_i}^*$ be the amount paid to charity π_{c_i} in some optimal solution. If we change this amount to π'_{c_i} and leave everything else unchanged, this will only affect the payment that we can extract from the bidders because of this particular charity, and the difference

in surplus will be $\sum_{b_j} u^i_j(\pi'_{c_i}) - u^i_j(\pi^*_{c_i}) - \pi'_{c_i} + \pi^*_{c_i}$. This expression is, of course, 0 if $\pi'_{c_i} = \pi^*_{c_i}$. But now notice that this expression is maximized as a function of π'_{c_i} by the decomposed solution for this charity (the terms without π'_{c_i} in them do not matter, and of course in the decomposed solution we always set $\pi_{b^i_j} = u^i_j(\pi_{c_i})$). It follows that if we change π_{c_i} to the decomposed solution, the change in surplus will be at least 0 (and the solution will still be valid). Thus, we can change the π_{c_i} one by one to the decomposed solution without ever losing any surplus. ■

Theorem 12 *When all bids are quasilinear and surplus is the objective, DONATION-CLEARING can be done in linear time.*

Proof: By Lemma 8, we can solve the problem separately for each charity. For charity c_i, this amounts to maximizing $(\sum_{b_j} u^i_j(\pi_{c_i})) - \pi_{c_i}$ as a function of π_{c_i}. Because all its terms are piecewise linear functions, this whole function is piecewise linear, and must be maximized at one of the points where it is nondifferentiable. It follows that we need only check all the points at which one of the terms is nondifferentiable. ■

Unfortunately, the decomposing lemma does not hold for payment maximization.

Proposition 1 *When the objective is payment maximization, even when bids are quasilinear, the solution obtained by decomposing the problem across charities is in general not optimal (even with concave bids).*

Proof: Consider a single bidder b_1 placing the following quasilinear bid over two charities c_1 and c_2: $u^1_1(\pi_{c_1}) = 2\pi_{c_i}$ for $0 \leq \pi_{c_i} \leq 1$, $u^1_1(\pi_{c_1}) = 2 + \frac{\pi_{c_i}-1}{4}$ otherwise; $u^2_1(\pi_{c_2}) = \frac{\pi_{c_i}}{2}$. The decomposed solution is $\pi_{c_1} = \frac{7}{3}$, $\pi_{c_2} = 0$, for a total donation of $\frac{7}{3}$. But the solution $\pi_{c_1} = 1$, $\pi_{c_2} = 2$ is also valid, for a total donation of $3 > \frac{7}{3}$. ■

In fact, when payment maximization is the objective, DONATION-CLEARING remains (weakly) NP-complete in general.

Theorem 13 *DONATION-CLEARING is (weakly) NP-complete when payment maximization is the objective, even when every bid is concerns only one charity (and has a step-function utility function for this charity), and is quasilinear.*

Proof: That the problem is in NP follows from the fact that the more general problem is in NP. To show NP-hardness, we reduce an arbitrary KNAPSACK instance (given by m pairs $(k_i, v_i)_{1 \leq i \leq m}$, a cost limit K, and a target value V), to the following DONATION-CLEARING instance. Let there be $m + 1$ charities, $c_0, c_1, \ldots, c_m$. Let there be one quasilinear bidder b_0 bidding $u^0_0(\pi_{c_0}) = 0$ for $0 \leq \pi_{c_0} \leq 1$, $u^0_0(\pi_{c_0}) = K + 1$ otherwise. Additionally, for each $1 \leq j \leq m$, let there be a bidder b_j bidding $u^j_j(\pi_{c_j}) = 0$ for $0 \leq \pi_{c_j} < k_i$, $u^j_j(\pi_{c_j}) = \epsilon v_i$ otherwise (where $\epsilon \sum_{1 \leq j \leq m} v_i < 1$). Let the target total amount donated be $K + 1 + \epsilon V$. We now show the two instances are equivalent.

First, suppose there exists a solution to the KNAPSACK instance, that is, a function $f : \{1,\ldots,m\} \rightarrow \{0,1\}$ so that $\sum_{i=1}^{m} f(i)k_i \leq K$ and $\sum_{i=1}^{m} f(i)v_i \geq V$. Then, let $\pi_{c_0} = 1 + \epsilon V + K - \sum_{i=1}^{m} f(i)k_i$, and for $i > 0$, $\pi_{c_i} = f(i)k_i$, for a total donated of $K + 1 + \epsilon V$. Because $1 + \epsilon V + K - \sum_{i=1}^{m} f(i)k_i \geq 1$, b_0's utility is $K + 1$. For $j > 0$, b_j's utility is $f(j)\epsilon v_j$, for a total utility of $\sum_{j=1}^{m} f(j)\epsilon v_j \geq \epsilon V$ for these m bidders. It follows that the total utility is at least the total amount donated, and the outcome is valid. So there exists a solution to the DONATION-CLEARING instance.

Now suppose there exists a solution to the DONATION-CLEARING instance. Let $f : \{1,\ldots,m\} \rightarrow \{0,1\}$ be given by $f(i) = 0$ if $\pi_{c_i} < k_i$, and $f(i) = 1$ otherwise. Because the total donated is at least $K+1+\epsilon V$, and the amount that is extractable from the bidders is at most $K+1+\sum_{j=1}^{m} f(j)\epsilon v_j$, it follows that $\sum_{j=1}^{m} f(j)v_j \geq V$. Also, because the total amount donated to charities 1 through m can be at most $K + \epsilon \sum_{1\leq j\leq m} v_i < K + 1$, it follows that $\sum_{j=1}^{m} f(j)k_i < K + 1$. Because the k_i are integers, this means $\sum_{j=1}^{m} f(j)k_i \leq K$. So there exists a solution to the KNAPSACK instance. ■

However, when the bids are also concave, a simple greedy clearing algorithm is optimal.

Theorem 14 *Given a DONATION-CLEARING instance with payment maximization as the objective where all bids are quasilinear and concave, consider the following algorithm. Start with* $\pi_{c_i} = 0$ *for all charities. Then, letting* $\gamma_{c_i} = \frac{d\sum_{b_j} u_j^i(\pi_{c_i})}{d\pi_{c_i}}$ *(at nondifferentiable points, these derivatives should be taken from the right), increase* $\pi_{c_i^*}$ *(where* $c_i^* \in \arg\max_{c_i} \gamma_{c_i}$*), until either* $\gamma_{c_i^*}$ *is no longer the highest (in which case, recompute* c_i^* *and start increasing the corresponding payment), or* $\sum_{b_j} u_j = \sum_{c_i} \pi_{c_i}$ *and* $\gamma_{c_i^*} < 1$*. Finally, let* $\pi_{b_j} = u_j$*.*

Proof: The outcome is valid because everyone pays exactly what she is willing to, and because there is no budget deficit: $\sum_{b_j} \pi_{b_j} = \sum_{b_j} u_j = \sum_{c_i} \pi_{c_i}$. To show optimality, let $\pi^*_{c_i}$ be the amount paid to charity c_i in some optimal solution, and let π'_{c_i} be the amount paid to charity i in the solution given by the greedy algorithm. We first observe that it is not possible that for any i, $\pi^*_{c_i} \geq \pi'_{c_i}$ with at least one of these inequalities being strict. This is because at the solution found by the greedy algorithm, $\gamma_{c_i^*}$ is less than 1; hence, using concavity, if $\pi^*_{c_i} > \pi'_{c_i}$, then $\int_{\pi'_{c_i}}^{\pi^*_{c_i}} \gamma_{c_i} d\pi_{c_i} < \pi^*_{c_i} - \pi'_{c_i}$. In other words, the additional payment that needs to be made to the charity is less than the additional payment that can be collected from the bidders because of this charity. Because the surplus at the greedy algorithm's solution is 0, it follows that if for any i, $\pi^*_{c_i} \geq \pi'_{c_i}$ with at least one of these inequalities being strict, the surplus at the optimal solution woud be negative, and hence the solution would not

be valid. Thus, either for all i, $\pi^*_{c_i} \leq \pi'_{c_i}$ (but in this case the greedy solution has at least as large a total payment as the optimal solution, and we are done); or there exist i, j such that $\pi^*_{c_i} > \pi'_{c_i}$ but $\pi^*_{c_j} < \pi'_{c_j}$. It cannot be the case that $\gamma_{c_i}(\pi'_{c_i}) > \gamma_{c_j}(\pi^*_{c_j})$, for then the greedy algorithm would have increased π_{c_i} beyond π'_{c_i} before increasing π_{c_j} beyond $\pi^*_{c_j}$. So, $\gamma_{c_i}(\pi'_{c_i}) \leq \gamma_{c_j}(\pi^*_{c_j})$. Because $\pi^*_{c_i} > \pi'_{c_i}$, and using concavity, if we decrease $\pi^*_{c_i}$ and simultaneously increase $\pi^*_{c_j}$ by the same amount, we will not decrease the total payment we can extract—while keeping the payment to be made to the charities the same. It follows this cannot make the solution worse or invalid. We can keep doing this until there is no longer a pair i, j such that $\pi^*_{c_i} > \pi'_{c_i}$ but $\pi^*_{c_j} < \pi_{c_j}$, and by the previous we know that for all i, $\pi^*_{c_i} \leq \pi'_{c_i}$—and hence the greedy solution is optimal. ■

(A similar greedy algorithm works when the objective is surplus and the bids are quasilinear and concave, with as only difference that we stop increasing the payments as soon as $\gamma_{c^*_i} < 1$.)

This concludes the part of this dissertation studying the complexity of the outcome optimization problem for expressive preference aggregation for donations to charities; we will study mechanism design aspects of this setting in the chapter after the next chapter, Section 5.2. In the next section, we study the complexity of the outcome optimization problem in more general settings with externalities.

3.4 Expressive preference aggregation in settings with externalities

In this section, we study the optimization problem for expressive preference aggregation in settings with externalities, as defined in Section 2.4. We study the following two computational problems. (Recall that a solution is feasible if no agent prefers the default outcome (all variables set to 0) to it.)

Definition 14 (FEASIBLE-CONCESSIONS) *We are given a concessions setting (as defined in Section 2.4). We are asked whether there exists a nontrivial feasible solution.*

Definition 15 (SW-MAXIMIZING-CONCESSIONS) *We are given a concessions setting (as defined in Section 2.4). We are asked to find a feasible solution that maximizes social welfare (the sum of the agents' utilities).*

The following shows that if the first problem is hard, the second problem is hard to approximate to any ratio.

Proposition 2 *Suppose that FEASIBLE-CONCESSIONS is NP-hard even under some constraints on the instance (but no constraint that prohibits adding another agent that derives positive utility from any nontrivial setting of the variables of the other agents). Then it is NP-hard to approximate SW-MAXIMIZING-CONCESSIONS to any positive ratio, even under the same constraints.*

Proof: We reduce an arbitrary FEASIBLE-CONCESSIONS instance to a SW-MAXIMIZING-CONCESSIONS instance that is identical, except that a single additional agent has been added that derives positive utility from any nontrivial setting of the variable(s) of the other agents, and

to whose variables the other agents are completely indifferent (they cannot derive any utility from the new agent's variable(s)). If the original instance has no nontrivial feasible solution, then neither does the new instance, and the maximal social welfare that can be obtained is 0. On the other hand, if the original instance has a nontrivial feasible solution, then the new instance has a feasible solution with positive social welfare: the exact same solution is still feasible, and the new agent will get positive utility (and the others, nonnegative utility). It follows that any algorithm that approximates SW-MAXIMIZING-CONCESSIONS to some positive ratio will return a social welfare of 0 if there is no solution to the FEASIBLE-CONCESSIONS problem, and positive social welfare if there is a solution—and thus the algorithm could be used to solve an NP-hard problem. ■

3.4.1 Hardness with positive and negative externalities

We first show that if we do not make the assumption of only negative externalities, then finding a feasible solution is NP-complete even when each agent controls only one variable. (In all the problems that we study, membership in NP is straightforward, so we just give the hardness proof.)

Theorem 15 *FEASIBLE-CONCESSIONS is NP-complete, even when all utility functions decompose (and all the components u_i^k are step functions), and each agent controls only one variable.*

Proof: We reduce an arbitrary SAT instance (given by variables V and clauses C) to the following FEASIBLE-CONCESSIONS instance. Let the set of agents be as follows. For each variable $v \in V$, let there be an agent a_v, controlling a single variable x_{a_v}. Also, for every clause $c \in C$, let there be an agent a_c, controlling a single variable x_{a_c}. Finally, let there be a single agent a_0 controlling x_{a_0}. Let all the utility functions decompose, as follows: For any $v \in V$, $u_{a_v}^{a_v}(x_{a_v}) = -\delta_{x_{a_v} \geq 1}$. For any $v \in V$, $u_{a_v}^{a_0}(x_{a_0}) = \delta_{x_{a_0} \geq 1}$. For any $c \in C$, $u_{a_c}^{a_c}(x_{a_c}) = (n(c) - 2|V|)\delta_{x_{a_c} \geq 1}$ where $n(c)$ is the number of variables that occur in c in negated form. For any $c \in C$, $u_{a_c}^{a_0}(x_{a_0}) = (2|V| - 1)\delta_{x_{a_0} \geq 1}$. For any $c \in C$ and $v \in V$ where $+v$ occurs in c, $u_{a_c}^{a_v}(x_{a_v}) = \delta_{x_{a_v} \geq 1}$. For any $c \in C$ and $v \in V$ where $-v$ occurs in c, $u_{a_c}^{a_v}(x_{a_v}) = -\delta_{x_{a_v} \geq 1}$. $u_{a_0}^{a_0}(x_{a_0}) = -|C|\delta_{x_{a_0} \geq 1}$. For any $c \in C$, $u_{a_0}^{a_c}(x_{a_c}) = \delta_{x_{a_c} \geq 1}$. All the other functions are 0 everywhere. We proceed to show that the instances are equivalent.

First suppose there exists a solution to the SAT instance. Then, let $x_{a_v} = 1$ if v is set to *true* in the solution, and $x_{a_v} = 0$ if v is set to *false* in the solution. Let $x_{a_c} = 1$ for all $c \in C$, and let $x_{a_0} = 1$. Then, the utility of every a_v is at least $-1+1 = 0$. Also, the utility of a_0 is $-|C|+|C| = 0$. And, the utility of every a_c is $n(c) - 2|V| + 2|V| - 1 + pt(c) - nt(c) = n(c) - 1 + pt(c) - nt(c)$, where $pt(c)$ is the number of variables that occur positively in c and are set to *true*, and $nt(c)$ is the number of variables that occur negatively in c and are set to *true*. Of course, $pt(c) \geq 0$ and $-nt(c) \geq -n(c)$; and if at least one of the variables that occur positively in c is set to *true*, or at least one of the variables that occur negatively in c is set to *false*, then $pt(c) - nt(c) \geq -n(c) + 1$, so that the utility of a_c is at least $n(c) - 1 - n(c) + 1 = 0$. But this is always the case, because the assignment satisfies the clause. So there exists a solution to the FEASIBLE-CONCESSIONS instance.

Now suppose there exists a solution to the FEASIBLE-CONCESSIONS instance. If it were the case that $x_{a_0} < 1$, then for all the a_v we would have $x_{a_v} < 1$ (or a_v would have a negative

utility), and for all the a_c we would have $x_{a_c} < 1$ (because otherwise the highest utility possible for a_c is $n(c) - 2|V| < 0$, because all the x_{a_0} are below 1). So the solution would be trivial. It follows that $x_{a_0} \geq 1$. Thus, in order for a_0 to have nonnegative utility, it follows that for all $c \in C$, $x_{a_c} \geq 1$. Now, let v be set to *true* if $x_{a_v} = 1$, and to *false* if $x_{a_v} = 0$. So the utility of every a_c is $n(c) - 2|V| + 2|V| - 1 + pt(c) - nt(c) = n(c) - 1 + pt(c) - nt(c)$. In order for this to be nonnegative, we must have (for any c) that either $nt(c) < n(c)$ (at least one variable that occurs negatively in c is set to *false*) or $pt(c) > 0$ (at least one variable that occurs positively in c is set to *true*). So we have a satisfying assignment. ■

3.4.2 Hardness with only negative externalities

Next, we show that even if we do make the assumption of only negative externalities, then finding a feasible solution is still NP-complete, even when each agent controls at most two variables.

Theorem 16 *FEASIBLE-CONCESSIONS is NP-complete, even when there are only negative externalities, all utility functions decompose (and all the components are step functions), and each agent controls at most two variables.*

Proof: We reduce an arbitrary SAT instance to the following FEASIBLE-CONCESSIONS instance. Let the set of agents be as follows. For each variable $v \in V$, let there be an agent a_v, controlling variables $x^+_{a_v}$ and $x^-_{a_v}$. Also, for every clause $c \in C$, let there be an agent a_c, controlling a single variable x_{a_c}. Let all the utility functions decompose, as follows: For any $v \in V$, $u^{a_v,+}_{a_v}(x^+_{a_v}) = -|C|\delta_{x^+_{a_v} \geq 1}$, and $u^{a_v,-}_{a_v}(x^-_{a_v}) = -|C|\delta_{x^-_{a_v} \geq 1}$. For any $v \in V$ and $c \in C$, $u^{a_c}_{a_v}(x_{a_c}) = \delta_{x_{a_c} \geq 1}$. For any $c \in C$, $u^{a_c}_{a_c}(x_{a_c}) = -\delta_{x_{a_c} \geq 1}$. For any $c \in C$ and $v \in V$ where $+v$ occurs in c, $u^{a_v,+}_{a_c}(x^+_{a_v}) = \delta_{x^+_{a_v} \geq 1}$; and for any $c \in C$ and $v \in V$ where $-v$ occurs in c, $u^{a_v,-}_{a_c}(x^-_{a_v}) = \delta_{x^-_{a_v} \geq 1}$. All the other functions are 0 everywhere. We proceed to show that the instances are equivalent.

First suppose there exists a solution to the SAT instance. Then, let $x^+_{a_v} = 1$ if v is set to *true* in the solution, and $x^+_{a_v} = 0$ otherwise; and, let $x^-_{a_v} = 1$ if v is set to *false* in the solution, and $x^-_{a_v} = 0$ otherwise. Let $x_{a_c} = 1$ for all $c \in C$. Then, the utility of every a_v is $-|C| + |C| = 0$. Also, the utility of every a_c is at least $-1 + 1$ (because all clauses are satisfied in the solution, there is at least one $+v \in c$ with $x^+_{a_v} = 1$, or at least one $-v \in c$ with $x^-_{a_v} = 1$. So there exists a solution to the FEASIBLE-CONCESSIONS instance.

Now suppose there exists a solution to the FEASIBLE-CONCESSIONS instance. At least one of the $x^+_{a_v}$ or at least one of the $x^-_{a_v}$ must be set nontrivially (≥ 1), because otherwise no x_{a_c} can be set nontrivially. But this implies that for any clause $c \in C$, $x_{a_c} \geq 1$ (for otherwise the a_v with a nontrivial setting of its variables would have negative utility). So that none of the a_c have nonnegative utility, it must be the case that for any $c \in C$, either there is at least one $+v \in c$ with $x^+_{a_v} \geq 1$, or at least one $-v \in c$ with $x^-_{a_v} \geq 1$. Also, for no variable $v \in V$ can it be the case that both $x^+_{a_v} \geq 1$ and $x^-_{a_v} \geq 1$, as this would leave a_v with negative utility. But then, letting v be set to *true* if $x^+_{a_v} \geq 1$, and to *false* otherwise must satisfy every clause. So there exists a solution to the SAT instance. ■

3.4.3 An algorithm for the case of only negative externalities and one variable per agent

We have shown that with both positive and negative externalities, finding a feasible solution is hard even when each agent controls only one variable; and with only negative externalities, finding a feasible solution is hard even when each agent controls at most two variables. In this subsection we show that these results are, in a sense, tight, by giving an algorithm for the case where there are only negative externalities and each agent controls only one variable. Under some minimal assumptions, this algorithm will return (or converge to) the maximal feasible solution, that is, the solution in which the variables are set to values that are as large as possible. Although the setting for this algorithm may appear very restricted, it still allows for the solution of interesting problems. For example, consider governments negotiating over by how much to reduce their countries' carbon dioxide emissions, for the purpose of reducing global warming.

We will not require the assumption of decomposing utility functions in this subsection (except where stated). The following claim shows the sense in which the maximal solution is well-defined in the setting under discussion (there cannot be multiple maximal solutions, and under a continuity assumption, a maximal solution exists).

Theorem 17 *In a concessions setting with only negative externalities and in which each agent controls only one variable, let* $x_1, x_2, \ldots, x_n$ *and* $x'_1, x'_2, \ldots, x'_n$ *be two feasible solutions. Then* $\max\{x_1, x'_1\}, \max\{x_2, x'_2\}, \ldots, \max\{x_n, x'_n\}$ *is also a feasible solution. Moreover, if all the utility functions are continuous, then, letting* X_i *be the set of values for* x_i *that occur in some feasible solution,* $\sup(X_1), \sup(X_2), \ldots, \sup(X_n)$ *is also a feasible solution.*

Proof: For the first claim, we need to show that every agent i receives nonnegative utility in the proposed solution. Suppose without loss of generality that $x_i \geq x'_i$. Then, we have $u_i(\max\{x_1, x'_1\}, \max\{x_2, x'_2\}, \ldots, \max\{x_i, x'_i\}, \ldots, \max\{x_n, x'_n\}) = u_i(\max\{x_1, x'_1\}, \max\{x_2, x'_2\}, \ldots, x_i, \ldots, \max\{x_n, x'_n\}) \geq u_i(x_1, x_2, \ldots, x_i, \ldots, x_n)$, where the inequality stems from the fact that there are only negative externalities. But the last expression is nonnegative because the first solution is feasible.

For the second claim, we will find a sequence of feasible solutions that converges to the proposed solution. By continuity, any agent's utility at the limit point must be the limit of that agent's utility in the sequence of feasible solutions; and because these solutions are all feasible, this limit must be nonnegative. For each agent i, let $\{(x_1^{i,j}, x_2^{i,j}, \ldots, x_n^{i,j})\}_{j \in \mathbb{N}}$ be a sequence of feasible solutions with $\lim_{j \to \infty} x_i^{i,j} = \sup(X_i)$. By repeated application of the first claim, we have that (for any j) $\max_i\{x_1^{i,j}\}, \max_i\{x_2^{i,j}\}, \ldots, \max_i\{x_n^{i,j}\}$ is a feasible solution, giving us a new sequence of feasible solutions. Moreover, because this new sequence dominates every one of the original sequences, and for each agent i there is at least one original sequence where the ith element converges to $\sup(X_i)$, the sequence converges to the solution $\sup(X_1), \sup(X_2), \ldots, \sup(X_n)$. ■

We are now ready to present the algorithm. First, we give an informal description. The algorithm proceeds in stages: in each stage, for each agent, it eliminates all the values for that agent's variable that would result in a negative utility for that agent regardless of how the other agents set their variables (given that they use values that have not yet been eliminated).

ALGORITHM 1
1. **for** i := 1 **to** n $\{$
2. X_i^0 := $\mathbb{R}^{\geq 0}$ (alternatively, X_i^0 := $[0, M]$ where M is some upper bound) $\}$
3. t := 0
4. **repeat until** $((\forall i)\ X_i^t = X_i^{t-1})$ $\{$
5. $t := t + 1$
6. **for** i := 1 **to** n $\{$
7. X_i^t := $\{x_i \in X_i^{t-1} : \exists x_1 \in X_1^{t-1}, x_2 \in X_2^{t-1}, \ldots, x_{i-1} \in X_{i-1}^{t-1}, x_{i+1} \in X_{i+1}^{t-1}, \ldots, x_n \in X_n^{t-1} : u_i(x_1, x_2, \ldots, x_i, \ldots, x_n) \geq 0\}$ $\}$ $\}$

The set updates in step 7 of the algorithm are simple to perform, because all the X_i^t always take the form $[0, r]$, $[0, r)$, or $\mathbb{R}^{\geq 0}$ (because we are in a concessions setting), and in step 7 it never hurts to choose values for $x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$ that are as large as possible (because we have only negative externalities). Roughly, the goal of the algorithm is for $\sup(X_1^t), \sup(X_2^t), \ldots, \sup(X_n^t)$ to converge to the maximal feasible solution (that is, the feasible solution such that all of the variables are set to values at least as large as in any other feasible solution). We now show that the algorithm is sound, in the sense that it does not eliminate values of the x_i that occur in feasible solutions.

Theorem 18 *Suppose we are running Algorithm 1 in a concessions setting with only negative externalities where each agent controls only one variable. If for some t, $r \notin X_i^t$, then there is no feasible solution with x_i set to r.*

Proof: We will prove this by induction on t. For $t = 0$ the theorem is vacuously true. Now suppose we have proved it true for $t = k$; we will prove it true for $t = k + 1$. By the induction assumption, all feasible solutions lie within $X_1^k \times \ldots \times X_n^k$. But if $r \neq X_i^{k+1}$, this means exactly that there is no feasible solution in $X_1^k \times \ldots \times X_n^k$ with $x_i = r$. It follows there is no feasible solution with $x_i = r$ at all. ■

However, the algorithm is not complete, in the sense that (for some "unnatural" functions) it does not eliminate all the values of the x_i that do not occur in feasible solutions.

Proposition 3 *Suppose we are running Algorithm 1 in a concessions setting with only negative externalities where each agent controls only one variable. For some (discontinuous) utility functions (even ones that decompose), the algorithm will terminate with nontrivial X_i^t even though the only feasible solution is the zero solution.*

Proof: Consider the following symmetric example:

- $u_1^1(x_1) = -x_1$ for $x_1 < 1$, $u_1^1(x_1) = -2$ otherwise;
- $u_1^2(x_2) = (x_2)^2$ for $x_2 < 1$, $u_1^2(x_2) = 1$ otherwise;
- $u_2^1(x_1) = (x_1)^2$ for $x_1 < 1$, $u_2^1(x_1) = 1$ otherwise;
- $u_2^2(x_2) = -x_2$ for $x_2 < 1$, $u_2^2(x_2) = -2$ otherwise.

There is no solution with $x_1 \geq 1$ or $x_2 \geq 1$, because the corresponding agent's utility would definitely be negative. In order for agent 1 to have nonnegative utility we must have $(x_2)^2 \geq x_1$. Unless they are both zero, this implies $x_2 > x_1$. Similarly, in order for agent 2 to have nonnegative utility we must have $(x_1)^2 \geq x_2$, and unless they are both zero, this implies $x_1 > x_2$. It follows that the only solution is the zero solution. Unfortunately, in the algorithm, we first get $X_1^1 = X_2^1 = [0, 1)$; then also, we get $X_1^2 = X_2^2 = [0, 1)$ (for any $x_1 < 1$, we can set $x_2 = \sqrt{(}x_1) < 1$ and agent 1 will get utility 0, and similarly for agent 2). So the algorithm terminates. ■

However, if we make some reasonable assumptions on the utility functions (specifically, that they are either continuous or piecewise constant), then the algorithm is complete, in the sense that it will (eventually) remove any values of the x_i that are too large to occur in any feasible solution. Thus, the algorithm converges to the solution. We will present the case of continuous utility functions first.

Theorem 19 *Suppose we are running Algorithm 1 in a concessions setting with only negative externalities where each agent controls only one variable. Suppose that all the utility functions are continuous. Also, suppose that all the X_i^0 are initialized to $[0, M]$. Then, all the X_i^t are closed sets. Moreover, if the algorithm terminates after the tth iteration of the* **repeat** *loop, then $\sup(X_1^t), \sup(X_2^t), \ldots, \sup(X_n^t)$ is feasible, and it is the maximal solution. If the algorithm does not terminate, then $\lim_{t\to\infty} \sup(X_1^t), \lim_{t\to\infty} \sup(X_2^t), \ldots, \lim_{t\to\infty} \sup(X_n^t)$ is feasible, and it is the maximal solution.*

Proof: First we show that all the X_i^t are closed sets, by induction on t. For $t = 0$, the claim is true, because $[0, M]$ is a closed set. Now suppose they are all closed for $t = k$; we will show them to be closed for $t = k + 1$. In the step in the algorithm in which we set X_i^{k+1}, in the choice of $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$, we may as well always set each of these x_j to $\sup(X_j^k)$ (which is inside X_j^k because X_j^k is closed by the induction assumption), because this will maximize agent i's utility. It follows that $X_i^{k+1} = \{x_i : u_i(\sup(X_1^k), \ldots, \sup(X_{i-1}^k), x_i, \sup(X_{i+1}^k), \ldots, \sup(X_n^k)) \geq 0\}$. But because u_i is continuous, this set must be closed by elementary results from analysis.

Now we proceed to show the second claim. Because each X_i^t is closed, it follows that $\sup(X_i^t) \in X_i^t$. This implies that, for every agent i, there exist $x_1 \in X_1^{t-1}, x_2 \in X_2^{t-1}, \ldots, x_{i-1} \in X_{i-1}^{t-1}, x_{i+1} \in X_{i+1}^{t-1}, \ldots, x_n \in X_n^{t-1}$ such that $u_i(x_1, x_2, \ldots, \sup(X_i^t), \ldots, x_n) \geq 0$. Because for every agent i', $X_{i'}^t = X_{i'}^{t-1}$ (the algorithm terminated), this is equivalent to saying that there exist $x_1 \in X_1^t, x_2 \in X_2^t, \ldots, x_{i-1} \in X_{i-1}^t, x_{i+1} \in X_{i+1}^t, \ldots, x_n \in X_n^t$ such that $u_i(x_1, x_2, \ldots, \sup(X_i^t), \ldots, x_n) \geq 0$. Of course, for each of these $x_{i'}$, we have $x_{i'} \leq \sup(X_{i'}^t)$. Because there are only negative externalities, it follows that $u_i(\sup(X_1^t), \sup(X_2^t), \ldots, \sup(X_i^t), \ldots, \sup(X_n^t)) \geq u_i(x_1, x_2, \ldots, \sup(X_i^t), \ldots, x_n) \geq 0$. Thus, $\sup(X_1^t), \sup(X_2^t), \ldots, \sup(X_n^t)$ is feasible. It is also maximal by Theorem 18.

Finally, we prove the third claim. For any agent i, for any t, we have $u_i(\sup(X_1^{t-1}), \sup(X_2^{t-1}), \ldots, \lim_{t'\to\infty} \sup(X_i^{t'}), \ldots, \sup(X_n^{t-1})) \geq u_i(\sup(X_1^{t-1}), \sup(X_2^{t-1}), \ldots, \sup(X_i^t), \ldots, \sup(X_n^{t-1}))$ (because the X_i^t are decreasing in t, and we are in a concessions setting). The last expression evaluates to a nonnegative quantity, using the same reasoning as in the proof of the second claim with the fact that $\sup(X_i^t) \in X_i^t$. But then, by continuity, $0 \leq \lim_{t\to\infty}(u_i(\sup(X_1^{t-1}), \sup(X_2^{t-1}), \ldots, \lim_{t'\to\infty} \sup(X_i^{t'}), \ldots, \sup(X_n^{t-1}))) = u_i(\lim_{t\to\infty} \sup(X_1^{t-1}), \lim_{t\to\infty} \sup(X_2^{t-1}), \ldots, \lim_{t'\to\infty} \sup(X_i^{t'}), \ldots, \lim_{t\to\infty} \sup(X_n^{t-1})) = u_i(\lim_{t\to\infty} \sup(X_1^t),$

$\lim_{t\to\infty} \sup(X_2^t), \ldots, \lim_{t\to\infty} \sup(X_i^t), \ldots, \lim_{t\to\infty} \sup(X_n^t)$. It follows that $\lim_{t\to\infty} \sup(X_1^t)$, $\lim_{t\to\infty} \sup(X_2^t), \ldots, \lim_{t\to\infty} \sup(X_n^t))$ is feasible. It is also maximal by Theorem 18. ■

We observe that piecewise constant functions are not continuous, and thus Theorem 19 does not apply to the case where the utility functions are piecewise constant. Nevertheless, the algorithm works on such utility functions, and we can even prove that the number of iterations is linear in the number of pieces. There is one caveat: the way we have defined piecewise constant functions (as linear combinations of step functions $\delta_{x\geq a}$), the maximal solution is not well defined (the set of feasible points is never closed on the right, i.e. it does not include its least upper bound). To remedy this, call a feasible solution *quasi-maximal* if there is no feasible solution that is larger (that is, all the x_i are set to values that are at least as large) and that gives some agent a different utility (so it is maximal for all intents and purposes).

Theorem 20 *Suppose we are running Algorithm 1 in a concessions setting with only negative externalities where each agent controls only one variable. If all the utility functions decompose and all the components u_i^k are piecewise constant with finitely many steps (the range of the u_i^k is finite), then the algorithm will terminate after at most T iterations of the* **repeat** *loop, where T is the total number of steps in all the self-components u_i^i (i.e. the sum of the sizes of the ranges of these functions). Moreover, if the algorithm terminates after the tth iteration of the* **repeat** *loop, then any solution $(x_1, x_2, \ldots, x_n)$ with for all i, $x_i \in \arg\max_{x_i \in X_i^t} \sum_{j\neq i} u_j^i(x_i)$, is feasible and quasi-maximal.*

Proof: If for some i and t, $X_i^t \neq X_i^{t-1}$, it must be the case that for some value r in the range of u_i^i, the preimage of this value is in $X_i^{t-1} - X_i^t$ (it has just been eliminated from consideration). Informally, one of the steps of the function u_i^i has been eliminated from consideration. Because this must occur for at least one agent in every iteration of the **repeat** loop before termination, it follows that there can be at most T iterations before termination. Now, if the algorithm terminates after the tth iteration of the **repeat** loop, and a solution $(x_1, x_2, \ldots, x_n)$ with for all i, $x_i \in \arg\max_{x_i \in X_i^t} \sum_{j\neq i} u_j^i(x_i)$ is chosen, it follows that each agent derives as much utility from the other agents' variables as is possible with the sets X_i^t (because of the assumption of only negative externalities, any setting of a variable that maximizes the total utility for the other agents also maximizes the utility for each individual other agent). We know that for each agent i, there is at least some setting of the other agents' variables within the X_j^t that will give agent i enough utility to compensate for the setting of its own variable (by the definition of X_i^t and using the fact that $X_j^t = X_j^{t-1}$, as the algorithm has terminated); and thus it follows that the utility maximizing setting is also enough to make i's utility nonnegative. So the solution is feasible. It is also quasi-maximal by Theorem 18. ■

Algorithm 1 can be extended to cases where some agents control multiple variables, by interpreting x_i in the algorithm as the *vector* of agent i's variables (and initializing the X_i^0 as cross products of sets). However, the next proposition shows how this extension of Algorithm 1 fails.

Proposition 4 *Suppose we are running the extension of Algorithm 1 just described in a concessions setting with only negative externalities. When some agents control more than one variable, the algorithm may terminate with nontrivial X_i^t even though the only feasible solution is the zero solution*

(all variables set to 0), even when all of the utility functions decompose and all of the components $u_i^{k,j}$ are step functions (or continuous functions).

Proof: Let each of three agents control two variables, with utility functions as follows:

- $u_1^{1,1}(x_1^1) = -3\delta_{x_1^1 \geq 1}$
- $u_1^{1,2}(x_1^2) = -3\delta_{x_1^2 \geq 1}$
- $u_2^{2,1}(x_2^1) = -3\delta_{x_2^1 \geq 1}$
- $u_2^{2,2}(x_2^2) = -3\delta_{x_2^2 \geq 1}$
- $u_3^{3,1}(x_3^1) = -3\delta_{x_3^1 \geq 1}$
- $u_3^{3,2}(x_3^2) = -3\delta_{x_3^2 \geq 1}$
- $u_1^{2,1}(x_2^1) = 2\delta_{x_2^1 \geq 1}$
- $u_1^{3,1}(x_3^1) = 2\delta_{x_3^1 \geq 1}$
- $u_2^{1,1}(x_1^1) = 2\delta_{x_1^1 \geq 1}$
- $u_2^{3,2}(x_3^2) = 2\delta_{x_3^2 \geq 1}$
- $u_3^{1,2}(x_1^2) = 2\delta_{x_1^2 \geq 1}$
- $u_3^{2,2}(x_2^2) = 2\delta_{x_2^2 \geq 1}$

Increasing any one of the variables to a value of at least 1 will decrease the corresponding agent's utility by 3, and will raise only one other agent's utility, by 2. It follows that there is no feasible solution besides the zero solution, because any other solution will have negative social welfare (total utility), and hence at least one agent must have negative utility.

In the algorithm, after the first iteration, it becomes clear that no agent can set both its variables to values of at least 1 (because each agent can derive at most $4 < 6$ utility from the other agents' variables). Nevertheless, for any agent, it still appears possible at this stage to set either (but not both) of its variables to a value of at least 1. Unfortunately, in the next iteration, this still appears possible (because each of the other agents could set the variable that is beneficial to this agent to a value of at least 1, leading to a utility of $4 > 3$ for the agent). It follows that the algorithm gets stuck.

These utility functions are easily made continuous, while changing neither the algorithm's behavior on them nor the set of feasible solutions—for instance, by making each function linear on the interval $[0, 1]$. ■

In the next subsection, we discuss *maximizing social welfare* under the conditions under which we showed Algorithm 1 to be successful in finding the maximal solution.

3.4.4 Maximizing social welfare remains hard

In a concessions setting with only negative externalities where each agent controls only one variable, the algorithm we provided in the previous subsection returns the *maximal* feasible solution, in a linear number of rounds for utility functions that decompose into piecewise constant functions. However, this may not be the most desirable solution. For instance, we may be interested in the feasible solution with the highest social welfare (that is, the highest sum of the agents' utilities). In this subsection we show that finding this solution remains hard, even in the setting in which Algorithm 1 finds the maximal solution fast.

Theorem 21 *The decision variant of SW-MAXIMIZING-CONCESSIONS (does there exist a feasible solution with social welfare $\geq K$?) is NP-complete, even when there are only negative externalities, all utility functions decompose (and all the components u_i^k are step functions), and each agent controls only one variable.*

Proof: We reduce an arbitrary EXACT-COVER-BY-3-SETS instance (given by a set S and subsets $S_1, S_2, \ldots, S_q$ ($|S_i| = 3$) to cover S with, without any overlap) to the following SW-MAXIMIZING-CONCESSIONS instance. Let the set of agents be as follows. For every S_i there is an agent a_{S_i}. Also, for every element $s \in S$ there is an agent a_s. Every agent a controls a single variable x_a. Let all the utility functions decompose, as follows: For any S_i, $u_{a_{S_i}}^{a_{S_i}}(x_{a_{S_i}}) = -7\delta_{x_{a_{S_i}} \geq 1}$. For any S_i and for any s, $u_{a_{S_i}}^{a_s}(x_{a_s}) = 7\delta_{x_{a_s} \geq 1}$. For any s, $u_{a_s}^{a_s}(x_{a_s}) = -\delta_{x_{a_s} \geq 1}$. For any s and for any S_i with $s \in S_i$, $u_{a_s}^{a_{S_i}}(x_{a_{S_i}}) = \frac{1}{q(s)-1}\delta_{x_{a_{S_i}} \geq 1}$, where $q(s)$ is the number of sets S_i with $s \in S_i$. Let the target social welfare be $7q(|S|-1) + 7\frac{|S|}{3}$. All the other functions are 0 everywhere. We proceed to show that the two instances are equivalent. First, suppose there exists a solution to the EXACT-COVER-BY-3-SETS instance. Then, let $x_{a_{S_i}} = 0$ if S_i is in the cover, and $x_{a_{S_i}} = 1$ otherwise. For all s, let $x_s = 1$. Then a_{S_i} receives a utility of $7|S|$ if S_i is in the cover, and $7(|S|-1)$ otherwise. Furthermore, for all $s \in S$, a_s receives a utility of $(q(s)-1)\frac{1}{q(s)-1} - 1 = 0$ (because for exactly $q(s)-1$ of the $q(s)$ subsets S_i with s in it, the corresponding agent has its variable set to 1: the only exception is the subset S_i that contains s and is in the cover). It follows that all the agents receive nonnegative utility, and the total utility (social welfare) is $7q(|S|-1) + 7\frac{|S|}{3}$. So there exists a solution to the SW-MAXIMIZING-CONCESSIONS instance. Now, suppose that there exists a solution to the SW-MAXIMIZING-CONCESSIONS instance. We first observe that if for some $s \in S$, $x_{a_s} < 1$, the total utility (social welfare) can be at most $7q(|S|-1)+2|S| < 7q(|S|-1)+7\frac{|S|}{3}$ (because each a_{S_i} can receive at most $7(|S|-1)$, and each a_s can receive at most $q(s)\frac{1}{q(s)-1}$, and because $q(s) \geq 2$ this can be at most 2). So it must be the case that $x_{a_s} \geq 1$ for all $s \in S$. It follows that, in order for none of these a_s to have nonnegative utility, for every $s \in S$, there are at least $q(s) - 1$ subsets S_i with $x_{a_{S_i}} \geq 1$ and $s \in S_i$. In other words, for every $s \in S$, there is at most one subset S_i with $s \in S_i$ with $x_{a_{S_i}} < 1$. In other words again, the subsets S_i with $s \in S_i$ with $x_{a_{S_i}} < 1$ are disjoint (and so there are at most $\frac{|S|}{3}$ of them. However, if there were only $k \leq \frac{|S|}{3} - 1$ subsets S_i with $x_{a_{S_i}} < 1$, then the total utility (social welfare) can be at most $7q(|S|-1)+7k+|S|-3k$ (each a_{S_i} receives at least $7(|S|-1)$, and they receive no more unless they are among the k, in which case they receive an additional 7; and every a_s receives 0 unless it is in none of the k disjoint subsets S_i, in which case it will receive at most 1 (because $q(s) \geq 2$, so $\frac{1}{q(s)-1} \leq 1$)—but of course there can be at most $|S|-3k$

such agents). But $7q(|S|-1)+7k+|S|-3k \leq 7q(|S|-1)+|S|+4(\frac{|S|}{3}-1) = 7q(|S|-1)+7\frac{|S|}{3}-4$, which is less than the target. It follows there are exactly $\frac{|S|}{3}$ disjoint subsets S_i with $x_{a_{S_i}} < 1$—an exact cover. So there exists a solution to the EXACT-COVER-BY-3-SETS instance. ■

3.4.5 Hardness with only two agents

So far, we have not assumed any bound on the number of agents. A natural question to ask is whether such a bound makes the problem easier to solve. In this subsection, we show that the problem of finding a feasible solution in a concessions setting with only negative externalities remains NP-complete even with only two agents (when there is no restriction on how many variables each agent controls).

Theorem 22 *FEASIBLE-CONCESSIONS is NP-complete, even when there are only two agents, there are only negative externalities, and all utility functions decompose (and all the components $u_i^{k,j}$ are step functions).*

Proof: We reduce an arbitrary KNAPSACK instance (given by r pairs (c_i, v_i), a cost constraint C and a value objective V) to the following FEASIBLE-CONCESSIONS instance with two agents. Agent 1 controls only one variable, x_1^1. Agent 2 controls r variables, $x_2^1, x_2^2, \ldots, x_2^r$. Agent 1's utility function is $u_1(x_1^1, x_2^1, x_2^2, \ldots, x_2^r) = -V\delta_{x_1^1 \geq 1} + \sum_{j=1}^{r} v_j \delta_{x_2^j \geq 1}$. Agent 2's utility function is $u_2(x_1^1, x_2^1, x_2^2, \ldots, x_2^r) = C\delta_{x_1^1 \geq 1} - \sum_{j=1}^{r} c_j \delta_{x_2^j \geq 1}$. We proceed to show that the instances are equivalent.

Suppose there is a solution to the KNAPSACK instance, that is, a subset $S \subseteq N$ such that $\sum_{j \in S} c_i \leq C$ and $\sum_{j \in S} v_i \geq V$. Then, let $x_1^1 = 1$, and for any $1 \leq j \leq r$, let $x_2^j = \delta_{j \in S}$. Then $u_1(x_1^1, x_2^1, x_2^2, \ldots, x_2^r) = -V + \sum_{j \in S} v_j \geq 0$. Also, $u_2(x_1^1, x_2^1, x_2^2, \ldots, x_2^r) = C - \sum_{j \in S} c_j \geq 0$. So there is a solution to the FEASIBLE-CONCESSIONS instance.

Now suppose there is a solution to the FEASIBLE-CONCESSIONS instance, that is, a nonzero setting of the variables $(x_1^1, x_2^1, x_2^2, \ldots, x_2^r)$ such that $u_1(x_1^1, x_2^1, x_2^2, \ldots, x_2^r) \geq 0$ and $u_2(x_1^1, x_2^1, x_2^2, \ldots, x_2^r) \geq 0$. If it were the case that $x_1^1 < 1$, then either all of agent 2's variables are set smaller than 1 (in which case x_1^1 must be nonzero and agent 1 gets negative utility), or at least one of agent 2's variables is nonzero (in which case agent 2 gets negative utility because the setting of x_1^1 is worthless to it). It follows that $x_1^1 \geq 1$. Thus, in order for agent 1 to get nonnegative utility, we must have $\sum_{j=1}^{r} v_j \delta_{x_2^j \geq 1} \geq V$. Let $S = \{j : x_2^j \geq 1\}$. Then it follows that $\sum_{j \in S} v_j \geq V$. Also, in order for agent 2 to get nonnegative utility, we must have $\sum_{j \in S} c_j \sum_{j=1}^{r} c_j \delta_{x_2^j \geq 1} \leq C$. So there is a solution to the KNAPSACK instance. ■

3.4.6 A special case that can be solved to optimality using linear programming

Finally, in this subsection, we demonstrate a special case in which we can find the feasible outcome that maximizes social welfare (or any other linear objective) in polynomial time, using linear programming. (Linear programs can be solved in polynomial time [Khachiyan, 1979].) The special case is the one in which all the utility functions decompose into piecewise linear, concave components. For this result we will need no additional assumptions (no bounds on the number of agents or variables per agent, *etc.*).

Theorem 23 *If all of the utility functions decompose, and all of the components $u_i^{k,j}$ are piecewise linear and concave, then SW-MAXIMIZING-CONCESSIONS can be solved in polynomial time using linear programming.*

Proof: Let the variables of the linear program be the x_i^j and the $u_i^{k,j}$. We use the following linear constraints: (1) For any i, we require $\sum_{k=1}^{n}\sum_{j=1}^{m_k} u_i^{k,j} \geq 0$; (2) For any i, k, j, for any linear function $l(x_k^j)$ that coincides with one of the segments of the function $u_i^{k,j}(x_k^j)$, we require $u_i^{k,j} \leq l(x_k^j)$.

The key observation is that for any value of x_k^j, the constraints allow one to set the variable $u_i^{k,j}$ to the value $u_i^{k,j}(x_k^j)$, but no larger: because the function $u_i^{k,j}(x_k^j)$ is concave, only the constraint corresponding to the segment that x_k^j is on is binding, and the constraints corresponding to other segments are not violated.

For the linear program's objective, we use $\sum_{i=1}^{n}\sum_{k=1}^{n}\sum_{j=1}^{m_k} u_i^{k,j}$, which is the social welfare. ∎

3.5 Summary

In this chapter, we studied the complexity of the outcome optimization problem for the four settings introduced in Chapter 2. While most voting rules are easy to execute, a few are not, including the Slater and Kemeny rules. In Section 3.1, we gave a powerful preprocessing technique for computing Slater rankings, showing that if a subset of the candidates consists of similar candidates, this subset can be solved recursively. We also gave an efficient algorithm for *finding* such a set of similar candidates, and provided experimental results showing the effectiveness of this preprocessing technique. Finally, we used the technique of similar sets to show that computing an optimal Slater ranking is NP-hard, even in the absence of pairwise ties.

In Section 3.2, we turned to the winner determination problem in combinatorial auctions. We studied the setting where there is a graph (with some desired property), with the items as vertices, and every bid bids on a connected set of items. Two computational problems arise: 1) clearing the auction when given the item graph, and 2) constructing an item graph (if one exists) with the desired property. We showed that given an item graph *with bounded treewidth*, the clearing problem can be solved in polynomial time (and every combinatorial auction instance has some treewidth; the complexity is exponential in only that parameter). We then gave an algorithm for constructing an item tree (treewidth 1) if such a tree exists. We showed why this algorithm does not work for treewidth greater than 1, but left open whether item graphs of (say) treewidth 2 can be constructed

in polynomial time (although we did show that finding the item graph with the fewest edges is NP-complete (even when a graph of treewidth 2 exists). We showed that the problems become hard if a bid is allowed to have more than one connected component.

In Section 3.3, we studied the outcome optimization problem for the setting of expressive negotation over donations to charities. We showed that this problem is NP-complete to approximate to any ratio even in very restricted settings. Subsequently, we gave a mixed integer program formulation of the clearing problem, and show that for concave bids, the program reduces to a linear program. We then showed that the clearing problem for a subclass of concave bids is at least as hard as a linear feasibility problem. Subsequently, we showed that the clearing problem is much easier when bids are quasilinear—for surplus, the problem decomposes across charities, and for payment maximization, a greedy approach is optimal if the bids are concave (although this latter problem is weakly NP-complete when the bids are not concave).

Finally, in Section 3.4, we studied the outcome optimization problem for the setting of expressive negotation in settings with externalities. The following table gives a summary of our results in that domain.

Restriction	**Complexity**
one variable per agent	NP-complete to find nontrivial feasible solution
negative externalities; two variables per agent	NP-complete to find nontrivial feasible solution
negative externalities; one variable per agent	Algorithm 1 finds maximal feasible solution (linear time for utilities that decompose into piecewise constant functions); NP-complete to find social-welfare maximizing solution
negative externalities; two agents	NP-complete to find nontrivial feasible solution
utilities decompose; components piecewise linear, concave	linear programming finds social welfare maximizing solution

Complexity of finding solutions in concessions settings. All of the hardness results hold even if the utility functions decompose into step functions.

One issue that we have not yet considered is that the agents will report their preferences *strategically*, that is, they will report them truthfully if and only if it is in their best interest to do so. This will be addressed in the deeper levels of the hierarchy, which will be the focus of the remainder of this dissertation. To prepare us for this, the next chapter reviews some basic concepts and results from *mechanism design*, which is the study of creating preference aggregation methods that are robust to this strategic behavior.