Combinatorial Auctions with Structured Item Graphs

Vincent Conitzer and **Jonathan Derryberry** and **Tuomas Sandholm**
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
{conitzer, jonderry, sandholm}@cs.cmu.edu

Abstract

Combinatorial auctions (CAs) are important mechanisms for allocating interrelated items. Unfortunately, winner determination is NP-complete unless there is special structure. We study 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. 1 was previously solved for the case of a tree or a cycle, and 2 for the case of a line graph or a cycle.

We generalize the first result by showing that given an item graph *with bounded treewidth*, the clearing problem can be solved in polynomial time (and every CA instance has some treewidth; the complexity is exponential in only that parameter). We then give an algorithm for constructing an item tree (treewidth 1) if such a tree exists, thus closing a recognized open problem. We show why this algorithm does not work for treewidth greater than 1, but leave open whether item graphs of (say) treewidth 2 can be constructed in polynomial time. We show that finding the item graph with the fewest edges is NP-complete (even when a graph of treewidth 2 exists). Finally, we study how the results change if a bid is allowed to have more than one connected component. Even for line graphs, we show that clearing is hard even with 2 components, and constructing the line graph is hard even with 5.

Introduction

Combinatorial auctions (CAs), where bidders can bid on *bundles* of items, have emerged as key mechanisms for allocating goods, tasks, resources, etc., both in computer science and economics. They are desirable when a bidder's valuation of a bundle might not equal the sum of his valuations of the individual items contained in the bundle.

Unfortunately, determining the winners in a general CA is NP-complete (Rothkopf, Pekeč, & Harstad 1998). Three main approaches have been pursued to address this: 1) designing optimal search algorithms that are often fast (e.g., (Sandholm 2002; Sandholm *et al.* 2001; Fujishima, Leyton-Brown, & Shoham 1999; Boutilier 2002)), but require exponential time in the worst case (unless P = NP), 2) designing approximation algorithms (e.g., (Hoos & Boutilier 2000; Zurel & Nisan 2001; van Hoesel & Müller 2001))—but unfortunately no polytime algorithm can guarantee an approximation (unless ZPP = NP) (Sandholm 2002), and 3) designing optimal polytime algorithms for restricted classes of CAs (e.g., (Rothkopf, Pekeč, & Harstad 1998; Tennenholtz 2000; Penn & Tennenholtz 2000; Sandholm & Suri 2003)).

This paper 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).

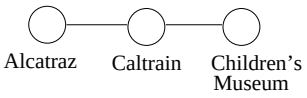
Consider graphs with the auction's items as vertices, which have the property that for any bid, the items occurring in it constitute a connected set in the graph. (For instance, the fully connected graph (with an edge between every pair of items) always has this property.) Such graphs can have potentially 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 has been solved for the special case where the graph is a tree or a cycle (Sandholm & Suri 2003); 2 has been solved for the special case where the graph is a line (Korte & Mohring 1989) or a cycle (Eschen & 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 & 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.

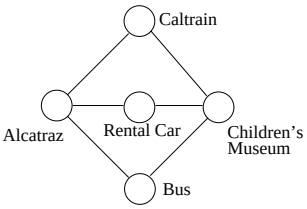
Also, there are two pure uses for graph structures which make questions 1 and 2 easy. First, the auctioneer can de-

cide 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, 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. Supposing (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 following line graph (because its only disconnected set is the one we just ruled out).



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 following is a valid item graph (because we just ruled out all its disconnected sets).



This 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 the paper 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 & 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 $k = 5$.

Item graphs

We first formally define item graphs.

Definition 1 *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.

Clearing with bounded treewidth item graphs

In this section, 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 & Suri 2003) which shows polynomial time clearability when the item graph is a tree (treewidth = 1).[1] 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 2 *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* $\max_{v \in T} |I_v| - 1$.

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, Corneil, & Proskurowski 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 1 *If $X \subseteq I$ is a connected set in G, then $\{v \in T : I_v \cap X \neq \{\}\}$ is connected in T.*

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

We are now ready to present our first result.

Theorem 1 *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. (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 $g : \{i \in I_v : f(i) = 0\} \rightarrow B \cup \{0\}$,[2] 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 $r(v, f)$.

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.

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

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 1, 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. ■

Constructing a valid item tree

So far we discussed question 1: how to clear the auction *given* a valid item graph. In this section, 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 & 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 section. This theorem shows how to give a tree that "minimally violates" the requirement of an item tree 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 2 *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 $A \leftarrow$ An $|I| \times |I|$ matrix of 0s
2 **for each** b **in** B
3 **do for each** i **in** $items(b)$
4 **do for each** $j \neq i$ **in** $items(b)$
5 **do** $A(i, j) \leftarrow A(i, j) + 1$
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, Leiserson, & Rivest 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. ∎

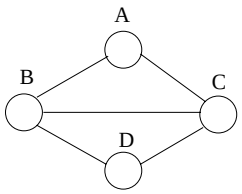
In particular, if an item tree exists for the given bids, then in that tree, each bid bids on only one connected component, so the summation in Theorem 2 is equal to the number of bids. Because each term in the summation must always be greater than or equal to 1, this tree minimizes the summation. Thus, MAKETREE will return a tree for which the summation in Theorem 2 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, and solve the rest of the problem using dynamic programming as in (Sandholm & 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.

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 following graph 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 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 section, we solve a related question.

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 polyno-

mial time (and P≠NP), the algorithm for doing so cannot be used to get the fewest edges—unlike the case of treewidth 1.

Theorem 3 *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.

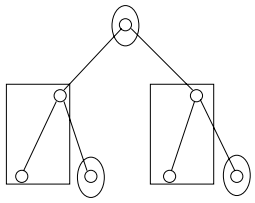
First, suppose there exists a solution to the 3SAT instance. Then, let there be an edge between any two items which constitute a bid by themselves; additionally, let there be an edge between i_{+v} and i_1 whenever v is set to *true* in the SAT solution, and an edge between i_{-v} and i_1 whenever v 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. ∎

Bids on multiple connected sets

In this section, 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. The following figure 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.)



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 4 *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 4 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 section, we show that unfortunately, detecting whether a valid line graph exists with multiple (5) components is also NP-complete.

Lemma 2 *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 3 *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: We omit the proof because of space constraint. ■

Theorem 5 *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 2 and Lemma 3, 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. ■

Conclusions and future research

Combinatorial auctions (CAs) are important mechanisms for allocating interrelated items. Unfortunately, (even approximate) winner determination is NP-complete unless there is special structure. In this paper, 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. 1 was previously solved for the case of a tree or a cycle, and 2 for the case of a line graph or a cycle.

We generalized the first result by showing that given an item graph *with bounded treewidth*, the clearing problem can be solved in polynomial time (and every CA 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, thus closing a recognized open problem. We showed why this algorithm does not work for treewidth greater than 1, but leave open whether item graphs of (say) treewidth 2 can be constructed in polynomial time. We showed that finding the item graph with the fewest edges is NP-complete (even when a graph of treewidth 2 exists). Finally, we studied how the results change if a bid is allowed to have a few connected components (rather than just one). Even for line graphs, we showed that clearing is hard even with 2 components, and constructing the line graph is hard even with 5 components.

For future research, the most important specific question left open is whether item graphs of bounded treewidth w can be constructed in polynomial time (when they exist), for $w \geq 2$. Another question is whether a line graph can be constructed in polynomial time in the variant where a bid is allowed to have k components, when $k < 5$. Also, we can ask whether we can reduce the runtime bound of the tree detection algorithm. More open-ended future research includes doing experimental work with the algorithms presented here (for instance, comparing their running time to solvers such as CPLEX), and integrating these algorithms into search algorithms as subroutines, as discussed in the introduction.

Acknowledgements

This material is based upon work supported by the National Science Foundation under CAREER Award IRI-9703122, Grant IIS-9800994, ITR IIS-0081246, and ITR IIS-0121678. Discussions with Jason Hartline and Daniel Sleator were helpful in working out ideas about how to construct an item tree from a bid set.

References

Akcoglu, K.; Aspnes, J.; DasGupta, B.; and Kao, M. Y. 2002. Opportunity cost algorithms for combinatorial auctions. *Applied Optimization: Computational Methods in Decision-Making, Economics and Finance* 455–479.

Areborg, S.; Corneil, D. G.; and Proskurowski, A. 1987. Complexity of finding embeddings in a K-tree. *SIAM Journal on Algebraic and Discrete Methods* 8:277–284.

Boutilier, C. 2002. Solving concisely expressed combinatorial auction problems. In *AAAI-02*, 359–366.

Cormen, T. H.; Leiserson, C. E.; and Rivest, R. L. 1990. *Introduction to Algorithms*. MIT Press.

de Vries, S., and Vohra, R. 2003. Combinatorial auctions: A survey. *INFORMS Journal on Computing*.

Eschen, E. M., and Spinrad, J. 1993. An $O(n^2)$ algorithm for circular-arc graph recognition. In *SODA-93*, 128–137.

Fujishima, Y.; Leyton-Brown, K.; and Shoham, Y. 1999. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *IJCAI*, 548–553.

Hoos, H., and Boutilier, C. 2000. Solving combinatorial auctions using stochastic local search. In *AAAI*, 22–29.

Korte, N., and Mohring, R. H. 1989. An incremental linear-time algorithm for recognizing interval graphs. *SIAM Journal on Computing* 18(1):68–81.

Penn, M., and Tennenholtz, M. 2000. Constrained multi-object auctions and b-matching. *Information Processing Letters* 75(1–2):29–34.

Rothkopf, M. H.; Pekeč, A.; and Harstad, R. M. 1998. Computationally manageable combinatorial auctions. *Management Science* 44(8):1131–1147.

Sandholm, T., and Suri, S. 2003. BOB: Improved winner determination in combinatorial auctions and generalizations. *Artificial Intelligence* 145:33–58. Early version: Improved Algorithms for Optimal Winner Determination in Combinatorial Auctions and Generalizations. *AAAI-00*, pp. 90–97.

Sandholm, T.; Suri, S.; Gilpin, A.; and Levine, D. 2001. CABOB: A fast optimal algorithm for combinatorial auctions. In *IJCAI*, 1102–1108.

Sandholm, T. 2002. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence* 135:1–54. Conference version appeared at *IJCAI-99*, pp. 542–547.

Tennenholtz, M. 2000. Some tractable combinatorial auctions. In *AAAI-00*.

van Hoesel, S., and Müller, R. 2001. Optimization in electronic marketplaces: Examples from combinatorial auctions. *Netnomics* 3(1):23–33.

Zurel, E., and Nisan, N. 2001. An efficient approximate allocation algorithm for combinatorial auctions. In *Proceedings of the ACM Conference on Electronic Commerce (ACM-EC)*, 125–136.