DIMACS Technical Report 2000-27
September 2000

Opportunity Cost Algorithms for Combinatorial Auctions

by

Karhan Akcoglu [1]
Department of Computer Science
Yale University
New Haven, CT 06520-8285.
Email: karhan.akcoglu@yale.edu

James Aspnes [2]
Department of Computer Science
Yale University
New Haven, CT 06520-8285.
Email: aspnes-james@cs.yale.edu

Bhaskar DasGupta [3]
Department of Computer Science
Rutgers University
Camden, NJ 08102.
Email: bhaskar@crab.rutgers.edu

Ming-Yang Kao [4]
Department of Computer Science
Yale University
New Haven, CT 06520-8285.
Email: kao-ming-yang@cs.yale.edu

[1]Supported in part by NSF Grant CCR-9896165.
[2]Supported in part by NSF grant CCR-9820888.
[3]Supported in part by NSF Grants CCR 99-06106, CCR 91-19999 and CCR-9800086.
[4]Supported in part by NSF Grant CCR-9531028.

ABSTRACT

Two general algorithms based on opportunity costs are given for approximating a revenue-maximizing set of bids an auctioneer should accept, in a *combinatorial auction* in which each bidder offers a price for some subset of the available goods and the auctioneer can only accept non-intersecting bids. Since this problem is difficult even to approximate in general, the algorithms are most useful when the bids are restricted to be connected node subsets of an underlying *object graph* that represents which objects are relevant to each other. The approximation ratios of the algorithms depend on structural properties of this graph and are small constants for many interesting families of object graphs. The running times of the algorithms are linear in the size of the *bid graph*, which describes the conflicts between bids. Extensions of the algorithms allow for efficient processing of additional constraints, such as budget constraints that associate bids with particular bidders and limit how many bids from a particular bidder can be accepted.

1 Introduction

Auctions are arguably the simplest and most popular means of price determination for multilateral trading without intermediary market makers [18, 25, 35, 9]. This paper considers the setting where there are (1) a group of competing *bidders* who bid to possess the auction *objects* and (2) an *auctioneer* who determines which bidders win which objects.

For the case of allocating a single object to one of many bidders, there is a wealth of literature on the following four widely used forms of auction [18, 25, 26]. In an *English auction* or *ascending bid auction*, the price of an object is successively raised until only one bidder remains and wins the object. In a *Dutch auction*, which is the converse of an English auction, an initial high price is subsequently lowered until a bidder accepts the current price. In a *first-price sealed-bid auction*, potential buyers submit sealed bids for an object. The highest bidder is awarded the object and pays the amount of her bid. In a *second-price sealed-bid auction*, the highest bidder wins the object but pays a price equal to the second-highest bid. In all these forms of auction, the auctioneer can determine the winning bid in time linear in the number of bids in a straightforward manner.

For the case of allocating multiple objects to multiple bidders[21, 17, 27, 12, 20, 8], *combinatorial auctions* are perhaps the most important form of auctions in the Internet Age, where bidders are increasingly software agents. Oftentimes a bid by an agent is a subset of the auction objects, and the agent needs the entire subset to complete a task. Different bids may share the same object, but the winning bids must not share any object [24]. Combinatorial auctions were first proposed by Rassenti *et al.* [29] as one-round mechanisms for airport time slot allocation. Banks *et al.* [3], DeMartini *et al.* [10], and Parkes and Ungar [28] formulated multiple-round mechanisms. It is in general NP-hard for the auctioneer to determine a set of winning bids of a combinatorial auction which maximizes the revenue of the auction. To address this computational difficulty, Rothkopf *et al.* [33] placed constraints on permissible bids. Lehmann *et al.* [22] and Fujishima *et al.* [11] considered approximation algorithms. Sandholm and Suri [34] designed *anytime* algorithms, which return a sequence of monotonically improving solutions that eventually converges to optimal.

In this paper, we propose a general framework to exploit topological structures of the bids to determine the winning bids with a provably good approximation ratio in linear time. The following discussion uses the sale of a car as a light-hearted example to explain our computational problems and key concepts.

Imagine that we are in the business of auctioning used cars. If we insist on selling each car as a unit, we can sell each car to the highest bidder. If we are willing to sell parts of the car, we can still sell each part to the highest bidder. But suppose that some bidders are only interested in buying several parts at once: Alice may not want to buy a tire unless she can get the wheel that goes with it, while Bob might only be interested in both rear wheels and the axle between them. How do we decide which of a set of conflicting bids to accept?

We will assume that our only goal is to maximize our total revenue. Then we can express this problem as a simple combinatorial optimization problem. We have some universe $\mathcal{O}$ of objects, and our buyers supply us with a set $\mathcal{A}$ of bids. The i-th bid consists of a subset A_i of $\mathcal{O}$ and a price p_i that the buyer is willing to pay for all of the objects in S_i. We would like to choose a collection of bids $\mathcal{B} \subseteq \mathcal{A}$ that yields the best possible total price while being *consistent*, in the sense that no two sets A_i and A_j in $\mathcal{B}$ overlap.

As the auctioneer, we can construct a *bid graph* G whose nodes are the bids and which has an edge between any two bids that share an object. Then, a set of consistent bids is simply an independent set in G, i.e., a set of nodes no two of which are connected by an edge. Each node is given a weight equal to the value of the bid it represents.

Sadly, this means that the problem of finding the most valuable consistent set of bids is a thinly-disguised version of the maximum weight independent set problem, which is not only NP-hard, but cannot be approximated to within a ratio $O(n^{1-\epsilon})$ for an n-node graph unless $P = NP$ [16].[1] Even for the simplest case when all node weights are one, the maximum weight independent set problem is NP-hard even when every vertex has degree at most d for any $d \geq 3$, and in fact cannot be approximated within a ratio of d^{ε} for some $\varepsilon > 0$ unless $P = NP$ [1].

[1] The fact that the bid graph is defined by the intersections of a collection of sets does not by itself help; any graph can be defined in this way.

The best known algorithm (for arbitrary d) achieves an approximation within a factor of $O(d/\log\log d)$ [15]. As a result, it seems hopeless if we model our combinatorial auction problem as an independent set problem unless we exploit the topological structure of the underlying bid graph.

Using ideas from the interval selection algorithm of Berman and DasGupta [7], we describe in Section 2 a linear-time improvement of the greedy algorithm, called the *opportunity cost algorithm*, for approximating maximum weight independent sets in ordered graphs.[2] We then describe a similar algorithm called the *local ratio opportunity cost algorithm*, based on ideas from the resource allocation algorithms of Bar-Noy *et al.*[4]. Both algorithms produce the same output, but the first has a more iterative structure and is easier to implement while the second has a more recursive structure and is easier to analyze.

These opportunity cost algorithms distinguish themselves from the straightforward greedy algorithm by taking into account the cost of excluding previously considered neighbors of a chosen node. Since this accounting requires propagating information only between neighbors, it increases the running time by at most a small constant factor, and yet in many cases produces a great improvement in the approximation ratio. The quality of the approximation depends on the local structure of the ordered input graph G. For each node v in G, we examine all of its successors (adjacent nodes that appear later in the ordering). The maximum size of any independent set among v and its successors is called the *directed local independence number at v*; we will write it as $\beta(v)$. The maximum value of $\beta(v)$ over all nodes in the graph will be written as $\beta(G)$,[3] and is the *directed local independence number* of G. Our algorithms approximate a maximum weight independent set to within a factor of β. By comparison, the greedy algorithm approximates a maximum weight independent set within a ratio of the maximum size of any independent subset of both the predecessors and the successors of any node, which in general can be much larger than β (see Section 2).

These new approximation results are useful only if we can exhibit interesting classes of graphs for which β is small. Graphs with $\beta = 1$ have been extensively studied in the graph theory literature; these are known as *chordal graphs*, and are precisely those graphs that can be represented as intersection graphs of subtrees of a forest, a class that includes both trees and interval graphs (more details are given in Section 3.1). We give additional results showing how to compute upper bounds on β for more general classes of graphs in Sections 3.2 and 3.3.

Among these tools for bounding β, one of particular interest to our hypothetical combinatorial auctioneer is the following generalization of the fact that intersection graphs of subtrees have β equal to one. Suppose that we have an *object graph* whose nodes are objects and in which an edge exists between any two objects that are relevant to each other in some way. (In the car example, there might be an edge between a wheel and its axle but not between a wheel and the hood ornament.) We demand that the objects in each bid be *germane* in the sense that they must form a connected node subset of the object graph. For many sparse object graphs, the intersection graph of all connected sets of vertices can be ordered so that a later set intersects an earlier set only if it intersects a "frontier set" that may be much smaller than the earlier set. It is immediate that β for the intersection graph is bounded by the size of the largest frontier set (more details are given in Lemma 8). Examples of such graphs are those of low treewidth (Theorem 9) and planar graphs (Corollary 10).

In Section 4 we show how to handle more complex constraints on acceptable sets of bids. We investigate scenarios where bids are grouped by bidder, and that each bidder is limited to some maximum number of winning bids (an *unweighted budget constraint*), or some maximum total cost of winning bids (a *weighted budget constraint*). By charging later bids an approximate opportunity cost for earlier bids in the same budget groups, we can solve these problems approximately with ratio $\beta+1$ with unweighted constraints and $2\beta+3$ for weighted constraints. The results for unweighted budget constraints can be further generalized for more complicated constraints.

Finally, in Section 5 we discuss some open problems suggested by the current work.

---

[2]These are graphs in which the nodes have been assigned an order; as we will see in Section 3.5, the choice of order for a given bid graph can have a large effect on how good an approximation we can get.

[3]Or simply β when G is clear from the context.

2 Simple combinatorial auctions

In this section, we describe our algorithms for approximating the maximum weight independent set problem, the opportunity cost algorithm and the local ratio opportunity cost algorithm . Both algorithms return the same approximation.

2.1 The opportunity cost algorithm

We will write $u \to v$ if $uv \in E$ and call u a *predecessor* of v and v a *successor* of u. The set of all predecessors of u will be written as $\delta^-(u)$ and the set of all successors as $\delta^+(u)$.

Given a directed acyclic graph $G_0 = (V_0, E_0)$ with weights weight(v) for each v in V, the opportunity cost algorithm, OPCOST, proceeds in two stages:

OC1 Traversing the nodes according to the topological order of G_0, compute a *value* value(u) for each node u. This value represents an estimate of the gain we expect by including u in the independent set; it is computed by taking the weight of u and subtracting off an *opportunity cost* consisting of the values of earlier positive-value nodes that conflict with u. Formally, let

$$\mathrm{value}(u) = \mathrm{weight}(u) - \sum_{v \to u} \max(0, \mathrm{value}(v)). \tag{1}$$

OC2 Processing the nodes in reverse topological order, add any node with non-negative value to the desired independent set $\mathcal{B}$ and discard its predecessors. Formally, let

$$\mathrm{select}(u) = [\mathrm{value}(u) \geq 0] \wedge \forall v \in \delta^+(u) : \neg\,\mathrm{select}(v). \tag{2}$$

The output of the algorithm is the set $\mathcal{B}$ defined as all u for which select(u) is true. This set $\mathcal{B}$ is clearly independent. In Section 2.3, we examine how close $\mathcal{B}$ is to optimal.

2.2 The local ratio opportunity cost algorithm

The *local ratio technique* can be used to recursively find approximate solutions to optimization problems over vectors in $\mathbb{R}^n$, subject to a set of feasibility constraints. It was originally developed by Bar-Yehuda and Even [6], and later extended by Bafna *et al.* [2], Bar-Yehuda [5], and Bar-Noy *et al.* [4].

Let $w \in \mathbb{R}^n$ be a *weight* vector. Let F be a set of feasibility constraints. A vector $x \in \mathbb{R}^n$ is a *feasible solution* to a given problem (F, w) if it satisfies all the constraints in F. The *w-weight* of a feasible solution x is defined to be the dot-product $w \cdot x$; for $r \geq 1$, x is an *r-approximation* with respect to (F, w) if $r \cdot w \cdot x \geq w \cdot x^*$, where x^* is a feasible solution maximizing the w-weight. An algorithm is said to have an *approximation ratio* of r if it always returns an r-approximate solution.

Lemma 1 (Local Ratio Lemma [6]) *Let F be a set of feasibility constraints. Let w, w_1 and w_2 be weight vectors such that $w = w_1 + w_2$. If x is an r-approximation with respect to (F, w_1) and (F, w_2), then x is an r-approximation with respect to (F, w).*

We now describe the local ratio opportunity cost algorithm, LR-OPCOST. Given a directed acyclic graph $G_0 = (V_0, E_0)$ with weights weight(v) for each $v \in V_0$, we pass $(G_0, \mathrm{weight}(\cdot))$ to the following recursive procedure. This procedure takes as input a graph G and a weight function w and proceeds as follows:

LR1 Delete all nodes in G with non-positive weight. Let this new graph be G_2.

LR2 If G_2 has no nodes, return the empty set.

LR3 Otherwise, select a node u with no predecessors in G_2, and decompose the weight function w as $w = w_1 + w_2$, where

$$w_1(v) = \begin{cases} w(u) & \text{if } v \in \{u\} \cup \delta^+(u), \\ 0 & \text{otherwise,} \end{cases}$$

and $w_2 = w - w_1$.

LR4 Solve the problem recursively using (G_2, w_2) as input. Let $\mathcal{B}_2$ be the approximation to a maximum weight independent set returned by this recursive call.

LR5 If $\mathcal{B}_2 \cup \{u\}$ is an independent set, return $\mathcal{B} = \mathcal{B}_2 \cup \{u\}$. Otherwise, return $\mathcal{B} = \mathcal{B}_2$.

Theorem 2 Opcost *and* LR-Opcost *return the same approximation to a maximum weight independent set.*

Proof: Consider a recursive call C of LR-Opcost. Let u be the node that is selected to be processed in step LR3. All of u's predecessors in the original graph G_0 have either been processed in a previous step LR3 or deleted in some step LR1. Therefore, the current weight of u, $w(u)$, as seen by the recursive call C, is just value(u), as defined in step OC1 of Opcost. Furthermore, we add node u to our independent set in step OC2 if and only if we add u to our independent set in step LR5. ∎

2.3 Approximation ratios

Theorem 3 Opcost *and* LR-Opcost *return a $\beta(G)$-approximation to a maximum weight independent set. Furthermore, there exist weights for which this bound is tight.*

Proof: We will prove the result for LR-Opcost. The full result follows from Theorem 2. Clearly, the returned set of nodes $\mathcal{B}$ is an independent set. By Lemma 1, we need only show that $\mathcal{B}$ is a β-approximation with respect to w_1 and w_2. We will prove this by induction on the recursion. The base case of the recursion is trivial, since there are no positive weight nodes.

For the inductive step, assume that $\mathcal{B}_2$ is a β-approximation with respect to w_2. Then $\mathcal{B}$ is also a β-approximation with respect to w_2 since $w_2(u) = 0$ and $\mathcal{B} \subset \mathcal{B}_2 \cup \{u\}$.

To show that $\mathcal{B}$ is a β-approximation with respect to w_1, we will derive an upper bound $\beta w(u)$ on the maximum w_1-weight independent set and a lower bound $w(u)$ on the w_1-weight of any *u-maximal* independent set of nodes. A u-maximal independent set of nodes either contains u or adding u to it violates the property that it is an independent set. Our w_1 performance bound is $\beta w(u)/w(u) = \beta$. Note that only u and its successor nodes will have a nonzero contribution to w_1-weight.

The total weight of a maximum w_1-weight independent set is at most $\beta(u)w(u) \leq \beta(G)w(u) = \beta w(u)$. The total weight of any u-maximal independent set is at least $w(u)$, since any such set contains at least one element of $u \cup \delta^+(u)$, and all such nodes are assigned weight $w(u)$. Since the algorithm always chooses a u-maximal set, its w_1 performance bound is β.

To show the bound is tight, pick some v that maximizes $\beta(v)$, and assign it weight 1 and all of its successors weight $1 - \epsilon$, where $\epsilon > 0$. Let every other node in G have weight 0. When we run Opcost, the value of v will be 1, the value of each of its successors will be $-\epsilon$, and the value of any other node is irrelevant because it has zero weight. Thus Opcost returns a set of total weight 1 but the maximum weight independent set has total weight at least $\beta(u) \cdot (1 - \epsilon)$. ∎

2.4 Running time

Theorem 4 *The running times of both* OPCOST *and* LR-OPCOST *are linear in the size of the input graph G_0.*

Proof: OPCOST computes value(v) for each node v in time proportional to its indegree, and computes select(v) for each node in time proportional to its outdegree, for a total time of $O(|V_0| + |E_0|)$. In the case of LR-OPCOST, a recursive call is made at most once for each node in the graph, and defining w_1 and w_2 in each call takes time proportional to the node's outdegree, for a total running time of $O(|V_0| + |E_0|)$. ∎

3 Properties of β

For any v, $\beta(v)$ is at most the larger of 1 or the outdegree of v. Thus, $\beta(G)$ is at most the larger of 1 or the maximum degree of G. In many cases we can use the structure of G to get a much better bound.

3.1 Graphs with $\beta = 1$

Graphs with orientations for which $\beta = 1$ can be characterized completely. These are the *chordal* graphs, also known as *triangulated* graphs or *rigid circuit* graphs. The defining property of a chordal graph is that no cycle of length 4 or more appears as an induced subgraph. A succinct discussion of these graphs, including a variety of characterizations as well as several examples of interesting families of chordal graphs, can be found in [14, pp. 280–281]. For our purposes the most useful of these characterizations are stated in the following lemma:

Lemma 5 *Let G be an undirected graph. Then the following properties of G are equivalent:*

1. *G is chordal.*

2. *G is the intersection graph of subtrees of a forest.*

3. *G has an ordering G' for which the successors of any node form a clique. Such an ordering is called a* perfect elimination ordering. *Restated in terms of β, G has an ordering G' for which $\beta(G') = 1$.*

Proof: See [14, pp. 280-281]. ∎

Chordal graphs can be recognized and ordered using a specialized version of breadth-first search in $O(|V|+|E|)$ time as shown by Rose *et al.* [32], and their maximum cardinality independent sets can be computed in $O(|V| + |E|)$ time as shown by Gavril [13]. Gavril's algorithm is essentially the same as step OC1 of the opportunity cost algorithm; it chooses all nodes with positive value and works because the sets $\{v : u \to v\}$ for each u in the independent set form a clique covering. However, this algorithm does not deal with weights.

Special cases of graphs with $\beta = 1$ include trees, interval graphs, and disjoint unions of cliques. The last are particularly nice:

Lemma 6 *Let G be a disjoint union of cliques. Then* every *orientation G' of G has $\beta(G') = 1$.*

Proof: For each u in G', $\delta^+(u)$ is a clique. ∎

3.2 Graphs with larger β values

For general graphs, we cannot compute β even approximately. However, we can bound the β values of many graphs using the tools in this section.

Lemma 7 *Let G be a directed graph.*

1. *If $G = G_1 \cup G_2$, then $\beta(G) \leq \beta(G_1) + \beta(G_2)$.*
2. *If G is a node-induced subgraph of H, then $\beta(G) \leq \beta(H)$.*

Proof: Let u be a node of G. Let $\delta^+(u)$, $\delta^+_1(u)$, $\delta^+_2(u)$, and $\delta^+_H(u)$ be the set of all successors of u in G, G_1, and G_2, respectively. Let A be any independent subset of $\delta^+(u)$. Then

1. $|A| \leq |A \cap \delta^+_1(u)| + |A \cap \delta^+_2(u)| \leq \beta(G_1) + \beta(G_2)$, and
2. A is an independent subset of $\delta^+_H(u)$, implying $|A| \leq \beta(H)$.

∎

Lemma 8 *Let G be the intersection graph of a set system $\mathcal{A}$ whose union is $\mathcal{O}$. Let G be ordered by an ordering $<$ such that for each $A \in \mathcal{A}$ there exists a "frontier set" $S_A \subseteq U$ of size at most k, so that if $A < B$ and $A \cap B \neq \emptyset$, then $S_A \cap B \neq \emptyset$. Then $\beta(G) \leq k$. (Note that S_A need not be contained in A.)*

Proof: Let $B_1, \ldots, B_l$ be some independent set of successors of A. Under the conditions of the lemma each B_i intersects S_A. But since the B_i do not themselves intersect, each must intersect S_A in a distinct element. Thus there are at most k of them. ∎

The converse of the lemma does not hold. Instead, its proof shows that the *clique covering number* $\overline{\chi}$ of $\delta^+(A)$ (defined as the minimum size of any set of cliques whose union is $\delta^+(A)$) is at most k, since the set of all B that intersect S_A at any particular element form a clique. Note that *any* directed acyclic graph in which $\overline{\chi}(\delta^+(v))$ is bounded can be represented as an intersection graph with small frontier sets as in Lemma 8,[4] in general the independence number of $\delta^+(v)$ may be smaller than the clique covering number.

When $\mathcal{A}$ consists of connected node subsets of some graph H, we can obtain good orderings of the intersection graph G of $\mathcal{A}$ by exploiting the structure of H.

We start by reviewing the definition of treewidth. A *tree decomposition* of an *undirected* graph $H = (V, E)$ consists of a tree T and a family of sets $\mathcal{V} = \{V_t\}$ where t ranges over nodes of T, satisfying the following three properties:

1. $\bigcup_{t \in T} V_t = V$.
2. For every edge uv in E, there is some V_t that contains both u and v.
3. If t_2 lies on the unique path from t_1 to t_3 in T, then $V_{t_1} \cap V_{t_3} \subseteq V_{t_2}$.

The *width* of a tree decomposition $(T, \mathcal{V})$ is $\max |V_t| - 1$. The *treewidth* $\mathrm{tw}(H)$ of a graph H is the smallest width of any tree decomposition of H.

Theorem 9 *If G is an intersection graph of connected node subsets $\mathcal{A}$ of some graph H with treewidth k, then there is an orientation G' of G with $\beta(G') \leq k + 1$. Given $\mathcal{A} = \{A_i\}$ and a tree decomposition $(T, \mathcal{V} = \{V_t\})$ of H, this orientation can be computed in time $O(\sum_i |A_i| + |T| + \sum_t |V_t|)$, which is linear in the size of the input.*

[4] The trick is to add a new common element to all members of each clique, and let S_A be the set of all such new elements for the cliques that cover $\delta^+(A)$.

Proof: Let $(T, \mathcal{V})$ be a tree decomposition of H with width k. We will use this tree decomposition to construct an ordering of the connected node subsets of H, with the property that if $A < B$ then either $A \cap B = \emptyset$ or B intersects some frontier set S_A with at most $k + 1$ elements. The full result then follows from Lemma 8.

Choose an arbitrary root r for T, and let $t_1 \geq t_2$ if t_1 is an ancestor of t_2 in the resulting rooted tree. Extend the resulting partial order to an arbitrary linear order. For each connected node subset A of H, let t_A be the greatest node in T for which V_{t_A} intersects A. Given two connected node subsets A and B of H, let $A < B$ if $t_A < t_B$ and extend the resulting partial order to any linear order.

Ordering T can be done in $O(|T|)$ time using depth first search. We can then compute and the maximum node in T containing each node of H in time $O(\sum_t V_t)$ by considering each V_t in order. The final step of ordering the A_i in the given set system $\mathcal{S}$ takes $O(\sum_i |A_i|)$ time, since we must examine each element of each A_i to find the maximum one. The total running time is thus linear in the size of the input.

Now suppose $A \leq B$ in this ordering. We will show that any such B intersects V_{t_A}, and thus that V_{t_A} is our desired frontier set S_A. There are two cases.

If $t_A = t_B$, we are done.

The case $t_A < t_B$ is more complicated. We will make heavy use of a lemma from [31], which concern the effect of removing some node t from T. Their Lemma 2.3 implies that if x, x' are not in V_t, then either x and x' are separated in H by V_t or x and x' are in the same branch (connected component) of $T - t$.

Let p be the parent of t_A (which exists because t_A is not the greatest element in the tree ordering). We have $A \cap V_p = \emptyset$ since $p > t_A$. Since A is a connected set, it cannot be separated without removing any of its nodes; thus by Lemma 2.3 every element of A is in the same branch of $T - p$, which consists precisely of the subtree of T rooted at t_A.

Now B contains at least one node x in the vertex set of an element of the subtree rooted at t_A, and at least one node x' in V_{t_B}, which is not in this subtree because $t_B > t_A$. So by Lemma 2.3 of [31], either one of x, x' is in V_{t_A} or B is separated by V_{t_A}. In the latter case B intersects V_{t_A} since B is also connected. ∎

Applying Theorem 9 to planar graphs gives:

Corollary 10 *If G is the intersection graph of a family $\mathcal{A}$ of connected node subsets of a planar graph H with n nodes, then there is an orientation G' of G with $\beta(G') = O(\sqrt{n})$. Given H, a data structure of size $O(n)$ can be precomputed in time $O(n \log n)$ that allows this orientation G' to be computed for any $\mathcal{A} = \{A_i\}$ in time $O(\sum_i |A_i|)$.*

Proof: Reed [30] gives a recursive $O(n \log n)$ algorithm for computing tree decompositions of constant-treewidth graphs based on a linear time algorithm for finding approximate separators for small node subsets. Replacing this separator-finding subroutine with the linear time algorithm of Lipton and Tarjan [23] gives an $O(n \log n)$ time algorithm for computing a tree decomposition of a planar graph. Since each separator has size at most $k = O(\sqrt{n})$, the resulting tree decomposition has width at most $4k = O(\sqrt{n})$ by Theorem 1 of [30].

Since all we need to compute a good ordering of $\mathcal{A}$ is the ordering of the n nodes, we can compute this ordering as described in the proof of Theorem 9 and represent it in $O(n)$ space by assigning each node an index in the range 1 to n. Ordering $\mathcal{A}$ then takes linear time as described in the proof of Theorem 9. ∎

3.3 Examples

Applying the results of Sections 3.1 and 3.2 gives:

1. A linear-time algorithm for finding a maximum weight independent set of an interval graph, since $\beta(G) = 1$ by Lemma 5, and since chordal graphs can be recognized and ordered in linear time using the work of Rose *et al.* [32].

While the maximum independent set problem is easily solved for this case (for example, by using the linear time interval graph recognition algorithm of Hsu and Ma [19] followed by a simple application of dynamic programming) this is an example of how our general method yields good algorithms as special cases.

2. As another special case, a 2-approximation algorithm for interval selection of Berman and DasGupta[7]. Here intervals are partitioned into groups and we must choose non-overlapping intervals with at most one per group. The bid graph G is of the form $G_1 \cup G_2$ where G_1 is an interval graph and G_2 is a disjoint union of cliques, one for each group. Thus $\beta(G) = 2$ by Lemmas 5, 6, and 7.

3. A 3-approximation algorithm for "double auction" interval selection where each interval has both a seller and a buyer, and at most one interval per seller or buyer may be selected. This is the same as the previous case except the graph is now $G_1 \cup G_2 \cup G_3$ where G_2 and G_3 are both disjoint unions of cliques.

4. In general, a mechanism for taking *any* bid graph with $\beta = k$ and adding up to m such unique-selection constraints to get a $(k + m)$-approximation algorithm by repeated applications of Lemmas 6 and 7. So for example we get a 3-approximation algorithm for maximum weight three-dimensional matching and a 4-approximation algorithm for auctioning off tracts of undeveloped land spanning intervals where each tract must be acceptable to a seller who provides it, a builder who will develop it, and a buyer who will ultimately purchase both the land and the buildings developed on it.

5. An algorithm to k-approximate a maximum weight independent set of any subgraph of a k-dimensional rectangular grid. Orient each edge to leave the point whose coordinates have a smaller sum, giving $\beta \leq k$.

6. A linear-time algorithm for 2-approximating a maximum weight independent set of the intersection graph of intervals on a cycle. This follows from Lemma 8: order connected node subsets by inclusion, extend to a linear order $\prec$, and observe that if $A \prec B$ and A intersects B then B intersects one of A's two endpoints.[5]

7. An algorithm for intersection graphs of bounded-height rectangles in a discrete 2D grid. Order the rectangles by their largest x-coordinate, and make the rightmost grid points of each rectangle be its frontier set in the sense of Lemma 8. If each rectangle is at most h tall, there are at most h grid points in each frontier. This generalizes in the obvious way to higher dimensions given bounds on all but one of the coordinates, in which case the approximation ratio becomes the product of the bounds.

3.4 Hardness of computing β

The difficulty of even approximating the independence number of a graph extends to the directed local independence number.

Theorem 11 *Any algorithm that can approximate $\beta(G)$ for an n-node directed acyclic graph G with a ratio of $f(n)$ can be used to approximate the size $\alpha(H)$ of a maximum independent set of an undirected n-node graph H with ratio $f(n+1)$. Thus by Håstad's bound on approximating a maximum clique [16], we cannot approximate β by $O(n^{1-\epsilon})$ unless $P = NP$.*

Proof: Given an undirected n-node graph H, construct an $(n + 1)$-node directed acyclic graph G by (a) directing the edges of H in any consistent order, and (b) adding a new source node s to H with edges from s to every node in H.

Let I be an independent set in H. Then every node in I is a successor of s in G, and furthermore these nodes are all independent. It follows that $\beta(G) \geq \beta(s) \geq \alpha(H)$.

Conversely, if I' is an independent set of successors of some node v in H, it cannot contain s (since s is not a successor of any node), and thus I' is also an independent set in H. So we have $\alpha(H) \geq \beta(G)$. ∎

[5] One can do better by breaking the cycle to reduce it to a standard interval graph problem (see, for example, the approach taken by [4]), but the 2-approximation shows how one can still do reasonably well with our general algorithms Opcost and LR-Opcost.

3.5 Effects of node ordering

The performance of the opportunity cost algorithm is strongly sensitive to the order in which the nodes are processed, as this affects the value of $\beta(u)$ for each node u. For many of the examples given in the Section 3.3, a good ordering is provided by the structure of the problem. But what happens in a general graph?

Theorem 12 *For any graph G with given weights, there exists an orientation G' of G for which both* OPCOST *and* LR-OPCOST *output a maximum independent set of G.*

Proof: Let A be any independent set in G. Choose the ordering so that all nodes in A precede all nodes not in A. Then for any $u \in A$, u has no predecessors in the oriented graph and $\text{value}(u) = \text{weight}(u)$.

Let A' be the independent set computed by the algorithm. If u is in A but not A', it must have a successor v in $A' - A$ with non-negative value. Since the value of each v is its weight less the weight of all its neighbors in A, the total weight of all elements of $A' - A$ must exceed the total weight of all elements in $A - A'$, and we have $\text{weight}(A') = \text{weight}(A' - A) + \text{weight}(A' \cap A) \geq \text{weight}(A - A') + \text{weight}(A' \cap A) = \text{weight}(A)$. ∎

In a sense what Theorem 12 shows is that finding a good ordering of a general graph is equivalent to solving the maximum weight independent set problem. This is not surprising since evaluating $\beta(u)$ for even a single node u requires solving this problem. It follows that to get small approximation ratios we really do need to exploit some special property of the given graph.

In the other direction, we can show that there exist orderings that are not very good:

Theorem 13 *If all nodes in a graph G have distinct weights, orienting G in order of decreasing weight causes* OPCOST *and* LR-OPCOST *to return the same independent set as the greedy algorithm.*

Proof: We will prove the result for OPCOST; by Theorem 2 the same result holds for LR-OPCOST.

Let π order the nodes in order of decreasing weight. Let us show by induction on π that if the greedy algorithm chooses a node v, then $\text{value}(v) = \text{weight}(v)$; but if the greedy algorithm does not choose v, then $\text{value}(v) < 0$. Suppose we are processing some node v and that this induction hypothesis holds for all nodes previously processed. If the greedy algorithm picks v, then all v's predecessors were not chosen and have negative value, and $\text{value}(v) = \text{weight}(v)$. If the greedy algorithm does not pick v, it is because it chose some $u \to v$; now $\text{value}(v) \leq \text{weight}(v) - \text{value}(u) = \text{weight}(v) - \text{weight}(u) < 0$.

Since the only nodes with non-negative weights are those chosen by the greedy algorithm, OPCOST selects them as its output. ∎

4 Auctions with budget constraints

Consider the following bidding scenarios:

1. A bidder whose car has broken down wants to buy either a new engine, a new car, or an umbrella and a taxi ride home, but doesn't particularly care which. However, she has no interest in winning more than one of these bids.

2. Another bidder wants to buy at most three 1968 Volkswagen Beetle hood ornaments, but she would like to bid on all that are available so as not to miss any.

3. Yet another bidder has only \$100 in cash, but would like to place multiple bids totaling more than \$100, with the understanding that she can only win bids up to her budget.

All of these are examples of *budget constraints*, in which bids in some group consume a common scarce resource. We would like to extend our algorithms to handle such constraints, which are natural in real-world bidding situations.

The first scenario is an example of a *1-of-n* constraint, where at most one of a set of n bids can be accepted. This special case can be handled by modifying G by forming a clique out of all bids in each set S_i; under the assumption that the S_i are disjoint, this increases β by at most 1 (using Lemmas 6 and 7). The second scenario depicts a more general *k-of-n* constraint. Such constraints are handled by extending our algorithms to account for the possible revenue loss from bids that cannot be selected because the budget constraint has been exceeded. Again, the approximation ratio rises by 1. We refer to both 1-of-n and k-of-n constraints as *unweighted budget constraints*, as each bid consumes a single unit of the budget.

Weighted budget constraints, exemplified by the third scenario, are more complicated. With such constraints, we must ensure that the sum of the weights of accepted bids in some group S is at most some bound b. A complication arises because a maximal allowed set of bids might only fill half of a budget limit. With some additional modifications to our algorithms, we get a performance bound of $2\beta + 3$.

4.1 Unweighted budget constraints

Suppose the bids are partitioned into groups $S_1, \ldots, S_r$ and that no more than k_i bids may be selected from S_i, for $1 \leq i \leq r$. For each bid u, let $g(u)$ denote the index of the group to which u belongs and let $S_u = S_{g(u)}$ and $k_u = k_{g(u)}$.

UNWEIGHTED-OPCOST is an extension of OPCOST to handle unweighted budget constraints. It has a similar two-step structure.

In the first step, like OC1, we traverse the nodes in topological order and compute a value for each node. We must extend the definition of value for each node to account for the *possible* revenue loss from previously processed bids that may not be selected in the second step because of the budget constraint:

$$\text{value}(u) = \text{weight}(u) - \sum_{v \to u} \max(0, \text{value}(v)) - \frac{1}{k_u} \cdot \sum_{v \in S_u - \{u\}, v < u} \max(0, \text{value}(v)), \tag{3}$$

where the notation $v < u$ means that v has already been processed (before u). Note that the inclusion of u in the set of winning bids does not necessarily preclude previously processed bids in S_u from also being selected—they may also be selected if the budget k_u allows. The coefficient $\frac{1}{k_u}$ scales the opportunity cost to account for this fact.

In the second step, like OC2, we traverse the bid graph in reverse topological order, selecting nodes of positive value whose addition to those already selected does not violate the independence or budget constraints.

UNWEIGHTED-LR-OPCOST solves the same problem using the local ratio technique. It follows the same structure as LR-OPCOST. We begin by deleting all non-positive weight nodes from the graph. If any nodes remain, we select a node u with no predecessors, and decompose the weight function into $w = w_1 + w_2$. This time, the decomposition must account for bids that are in the same budget group. We define

$$w_1(v) = \begin{cases} w(u) & \text{if } v \in \{u\} \cup \delta^+(u), \\ \frac{1}{k_u} w(u) & \text{if } v \in S_u - \{u\}, \\ 0 & \text{otherwise,} \end{cases}$$

and recursively solve the problem using w_2 as the weight function. After the recursive call, we must decide if we should add u to the set of winning bids $\mathcal{B}_2$. In LR-OPCOST, we added u to $\mathcal{B}_2$ if and only if $\mathcal{B}_2 \cup \{u\}$ was an independent set. In this algorithm, we must also ensure that the budget constraints are satisfied before adding u to $\mathcal{B}_2$. We say that a set of bids is *feasible* if they form an independent set and the budget constraints are satisfied.

Theorem 14 *Given a directed bid graph G, a partition of the nodes of G into nonempty subsets $S_1, \ldots, S_r$, and an unweighted budget constraint k_i for each S_i,*

1. Unweighted-Opcost *and* Unweighted-LR-Opcost *return the same approximation to a revenue maximizing set of bids.*

2. Unweighted-Opcost *and* Unweighted-LR-Opcost $(\beta(G)+1)$*-approximate an optimal set of bids.*

3. Unweighted-Opcost *and* Unweighted-LR-Opcost *run in time linear in the size of G.*

Proof: The proof that both algorithms return the same approximation is similar to the proof of Theorem 2.

The proof of the approximation ratio follows the same structure as the proof of Theorem 3. We prove the result for Unweighted-LR-Opcost. By Lemma 1, we need only show that the returned set of bids $\mathcal{B}$ is a $(\beta+1)$-approximation with respect to w_2 and w_1. We do this using induction on the recursion. The fact that $\mathcal{B}$ is a $(\beta+1)$-approximation with respect to w_2 follows trivially from the inductive assumption.

In the case of w_1, we will derive an upper bound U on the maximum w_1-weight of a set of feasible bids and a lower bound L on the w_1-weight of any *u-maximal* set of bids. A u-maximal set of bids either contains u or adding u to it would violate the feasibility constraints. In the case of a set of feasible bids, its total w_1-weight is at most $\beta(u)w(u) + \frac{k_u}{k_u}w(u) \leq w(u)(\beta+1) = U$, since the only nonzero contribution to w_1-weight comes from $\delta^+(u)$ and S_u. In the case of a u-maximal set of bids, if u cannot be added to the set, then either (1) a successor of u is already in the set, in which case the total w_1-weight is at least $w(u)$, or (2) the budget constraint is exceeded, in which case the total w_1-weight is at least $w(u)$. Therefore, the w_1-weight of these bids is at least $w(u)$ and the w_1 performance bound is

$$\frac{U}{L} = \frac{w(u)(\beta+1)}{w(u)} = \beta + 1.$$

The proof of the running time is similar to the proof of Theorem 4. All of the steps that Unweighted-Opcost and Unweighted-LR-Opcost share with Opcost and LR-Opcost take linear time. Unweighted-Opcost adds the cost of computing the last term in (3). Storing $\sum_{v \in S_i} \max(0, \text{value}(v))$ in a variable Δ_{S_i} for each S_i allows this term to be computed in time $O(1)$ for each node, with an additional $O(1)$ cost per node to update the appropriate S_i. The same technique allows budget constraints to be tested in $O(1)$ time per node during the second step. Thus the additional time is linear.

The corresponding modification to Unweighted-LR-Opcost similarly adds only linear time. Rather than updating the weight of each node v before each recursive call, we will compute the "current" weight of each node v as it is required, subtracting off the total weight Δ_{S_v} of all previously-processed nodes in S_v as in Unweighted-Opcost. ∎

4.2 Overlapping unweighted constraints

The analysis in Section 4.1 assumes that the budget constraints partition the bids. For some applications (e.g., bids involving matching up buyers with sellers), we may have overlapping constraints. Overlapping constraints may also be used to handle bids for identical items in limited supply, by grouping all bids asking for copies of the same item together. The algorithms described above can be generalized to handle overlapping constraints.

Suppose we have a family of r sets of bids $\mathcal{S} = \{S_1, \ldots, S_r\}$, that each bid appears in at most t of these sets, and that at most k_i bids may be accepted from set S_i.

In Overlapping-Unweighted-Opcost, when computing the value of a node u, we need to account for the possible revenue loss from nodes in each set that u belongs to:

$$\text{value}(u) = \text{weight}(u) - \sum_{v \to u} \max(0, \text{value}(v)) - \sum_{1 \leq i \leq r, u \in S_i} \left(\frac{1}{k^i} \sum_{v \in S_u, v < u} \max(0, \text{value}(v)) \right).$$

The rest of the algorithm is the same as UNWEIGHTED-OPCOST.

In OVERLAPPING-UNWEIGHTED-LR-OPCOST, the only change from UNWEIGHTED-LR-OPCOST is in the decomposition of the weight function. We decompose it as

$$w_1(v) = \begin{cases} w(u) & \text{if } v \in \{u\} \cup \delta^+(u), \\ \sum_{1 \le i \le r, u, v \in S_i} \frac{1}{k_i} w(u) & \text{if there exist } S_i \text{ containing both } u \text{ and } v, \\ 0 & \text{otherwise.} \end{cases}$$

Theorem 15 *Given a directed bid graph $G = (V, E)$, a family of nonempty node subsets $S_1, \ldots, S_r$, where each node appears in at most t of the S_i, and an unweighted budget constraint k_i for each S_i,*

1. OVERLAPPING-UNWEIGHTED-OPCOST *and* OVERLAPPING-UNWEIGHTED-LR-OPCOST *return the same approximation to a revenue maximizing set of bids.*
2. OVERLAPPING-UNWEIGHTED-OPCOST *and* OVERLAPPING-UNWEIGHTED-LR-OPCOST *$(\beta(G) + t)$-approximate an optimal set of bids.*
3. OVERLAPPING-UNWEIGHTED-OPCOST *and* OVERLAPPING-UNWEIGHTED-LR-OPCOST *run in time $O(|V|t + |E|)$*

Proof: Similar to the proof of Theorem 14. The additional $O(|V|t)$ term comes from having to apply up to t budget constraints to each node; since $\sum_i |S_i| \le |V|t$, this term also covers the cost of reading the S_i from the input and initializing the variables for each subset. ∎

4.3 Weighted budget constraints

Suppose that bids are partitioned into groups $S_1, \ldots, S_r$ and that the total value of the winning bids from group i can be no more than b_i. For each bid u, let $g(u)$ denote the index of the group to which u belongs and let $S_u = S_{g(u)}$ and $b_u = b_{g(u)}$.

This case is more complicated than the unweighted case. The difficulty arises when estimating a lower bound on the w_1-weight of a u-maximal set of bids S. If u cannot be added to the set because the budget constraint will be exceeded, the w_1-weight of S can be as small as ϵ, if $w_1(u) = b_u$.

We will describe changes required to LR-OPCOST to handle this case. Corresponding changes can be made to OPCOST. We will run variations of the algorithm twice, once for the *heavy* bids v with $w(v) > \frac{1}{2} b_v$ and once for the *light* bids v with $w(v) \le \frac{1}{2} b_v$. We then return the better of the two solutions.

In HEAVY-WEIGHTED-LR-OPCOST, we put an unweighted budget constraint of 1 on each bidder and run UNWEIGHTED-LR-OPCOST.

Lemma 16 HEAVY-WEIGHTED-LR-OPCOST *$(\beta + 1)$-approximates an optimal set of heavy bids.*

Proof: Since each heavy bid consumes more than half a bidder's budget, each bidder can win at most one bid. This is just a simple unweighted budget constraint and can be solved as described in Section 4.1 for a performance bound of $\beta + 1$. ∎

In LIGHT-WEIGHTED-LR-OPCOST, when decomposing the weight function, we set

$$w_1(v) = \begin{cases} w(u) & \text{if } v \in \{u\} \cup \delta^+(u), \\ \frac{2}{b_u} w(v) w(u) & \text{if } v \in S_u - \{u\}, \\ 0 & \text{otherwise.} \end{cases}$$

Before adding u to the winning set of bids $\mathcal{B}_2$, we must ensure that it does not conflict with other bids in $\mathcal{B}_2$ and that the weighted budget constraint is not violated. The rest of the algorithm is identical to LR-OPCOST.

Lemma 17 LIGHT-WEIGHTED-LR-OPCOST $(\beta+2)$*-approximates an optimal set of light bids.*

Proof: This proof uses the same structure and notation as the proof of Theorem 14. An upper bound U on the w_1-revenue of any feasible set of bids is $w(u)(\beta+2)$. With regards to a u-maximal set of bids, if u cannot be added to the set because the budget constraint b_u will be exceeded, the existing bids in the set must have weight at least $b_u/2$, since $w(u) \leq b_u/2$. A lower bound L on the w_1-revenue of any u-maximal set of bids is therefore $w(u)$. The performance bound of this algorithm is $\frac{U}{L} = \beta + 2$, as claimed. ∎

Theorem 18 *Given a directed bid graph G, a partition of the nodes of G into nonempty subsets $S_1, \ldots, S_r$, and a weighted budget constraint b_i for each S_i,*

1. WEIGHTED-OPCOST *and* WEIGHTED-LR-OPCOST *return the same approximation to a revenue maximizing set of bids.*
2. WEIGHTED-OPCOST *and* WEIGHTED-LR-OPCOST $(2\beta(G)+3)$*-approximate an optimal set of bids.*
3. WEIGHTED-OPCOST *and* WEIGHTED-LR-OPCOST *run in time linear in the size of G.*

Proof: The sum of the optimal revenues for the heavy and light bids is at least equal to the optimum revenue among all bids. From Lemmas 16 and 17, the better of the two solutions will be within a factor of $2\beta + 3$ of the optimum for the general problem.

For the running time, observe that decomposing the bids into heavy and light bids takes linear time, that HEAVY-WEIGHTED-OPCOST and HEAVY-WEIGHTED-LR-OPCOST are equivalent to UNWEIGHTED-OPCOST and UNWEIGHTED-LR-OPCOST and thus take linear time by Theorem 14, and that LIGHT-WEIGHTED-OPCOST and LIGHT-WEIGHTED-LR-OPCOST can be made to run in linear time using techniques similar to those used for UNWEIGHTED-OPCOST and UNWEIGHTED-LR-OPCOST. ∎

5 Further Research

This paper opens up several directions for further research. An immediate open problem is whether overlapping weighted budget constraints can be processed as efficiently as their unweighted counterparts are processed in Theorem 15.

It would be of importance to compare the performance of our algorithms and others in practice. The comparison could be conducted on simulations, but it would be more useful to analyze the performance on real auction data.

As the examples of car sales and land sales demonstrate, topological structures exist in actual bids. Another good example is the FCC auction of airwaves in 1994 and 1995 [24], where each trading area is an auction object, the trading areas form a plane graph, and bidders prefer to acquire contiguous trading areas. It would be useful to examine past auctions to determine whether similar connectivity structures exist and how such structures affect the computational complexity of bidding strategies and winner determination.

References

[1] N. ALON, U. FEIGE, A. WIGDERSON, AND D. ZUCKERMAN, *Derandomized graph products*, Computational Complexity, 5 (1995), pp. 60–75.

[2] V. BAFNA, P. BERMAN, AND T. FUJITO, *A 2-approximation algorithm for the undirected feedback vertex set problem*, SIAM Journal on Discrete Mathematics, 12 (1999), pp. 289–297.

[3] J. S. BANKS, J. O. LEDYARD, AND D. PORTER, *Allocating uncertain and unresponsive resources: An experimental approach*, The Rand Journal of Economics, 20 (1989), pp. 1–25.

[4] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. S. Naor, and B. Schieber, *A unified approach to approximating resource allocation and scheduling*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, 2000, pp. 735–744.

[5] R. Bar-Yehuda, *One for the price of two: a unified approach for approximating covering problems*, Algorithmica, 27 (2000), pp. 131–144.

[6] R. Bar-Yehuda and S. Even, *A local-ratio theorem for approximating the weighted vertex cover problem*, Annals of Discrete Mathematics, 25 (1985), pp. 27–45.

[7] P. Berman and B. DasGupta, *Improvements in throughput maximization for real-time scheduling*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, 2000, pp. 680–687.

[8] Y. Chen, M. Y. Kao, and H. I. Lu, *Optimal bid sequences for multiple-object auctions with unequal budgets*, in Lecture Notes in Computer Science: Proceedings of the 11th Annual International Symposium on Algorithms and Computation, D. T. Lee and S. H. Teng, eds., New York, NY, 2000, Springer-Verlag. To appear.

[9] S. H. Clearwater, ed., *Market-Based Control, a Paradigm for Distributed Resource Allocation*, World Scientific, River Ridge, NJ, 1996.

[10] C. DeMartini, A. M. Kwasnica, J. O. Ledyard, and D. Porter, *A new and improved design for multiple-object iterative auctions*, Tech. Report SSWP 1054, California Institute of Technology, 1999.

[11] Y. Fujishima, K. Leyton-Brown, and Y. Shoham, *Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches*, in Proceedings of the 16th International Joint Conference on Artificial Intelligence, 1999, pp. 548–553.

[12] I. Gale, *A multiple-object auction with superadditive values*, Economics Letters, 34 (1990), pp. 323–328.

[13] F. Gavril, *Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph*, SIAM Journal on Computing, 1 (1972), pp. 180–187.

[14] M. Grötschel, L. Lovász, and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, vol. 2 of Algorithms and Combinatorics, Springer-Verlag, 1988.

[15] M. M. Halldórsson and J. Radhakrishnan, *Improved approximations of independent sets in bounded-degree graphs via subgraph removal*, Nordic Journal of Computing, 1 (1994), pp. 475–482.

[16] J. Håstad, *Clique is hard to approximate within* $n^{1-\epsilon}$, Acta Mathematica, 182 (1999), pp. 105–142.

[17] D. B. Hausch, *Multi-object auctions: sequential vs. simultaneous sales*, Management Science. Journal of the Institute of Management Science. Application and Theory Series, 32 (1986), pp. 1599–1610, 1611–1612. With a comment by Michael H. Rothkopf, Elmer Dougherty and Marshall Rose.

[18] K. Hendricks and H. J. Paarsh, *A survey of recent empirical work concerning auctions*, Canadian Journal of Economics, 28 (1995), pp. 403–426.

[19] W.-L. Hsu and T.-H. Ma, *Fast and simple algorithms for recognizing chordal comparability graphs and interval graphs*, SIAM Journal on Computing, 28 (1999), pp. 1004–1020 (electronic).

[20] M. Y. Kao, J. F. Qi, and L. Tan, *Optimal bidding algorithms against cheating in multiple object auctions*, SIAM Journal on Computing, 28 (1999), pp. 955–969.

[21] V. Krishna and R. W. Rosenthal, *Simultaneous auctions with synergies*, Games and Economic Behavior, 17 (1996), pp. 1–31.

[22] D. Lehmann, L. I. O'Callaghan, and Y. Shoham, *Truth revelation in rapid approximately efficient combinatorial auctions*, in Proceedings of the 1st ACM Conference on Electronic Commerce, SIGecom, ACM Press, 1999, pp. 96–102.

[23] R. J. Lipton and R. E. Tarjan, *A separator theorem for planar graphs*, SIAM Journal of Applied Mathematics, 36 (1979), pp. 177–189.

[24] R. P. McAfee and J. McMillan, *Analyzing the airwaves auction*, Journal of Economic Perspectives, 10 (1996), pp. 159–175.

[25] J. McMillan and R. P. McAfee, *Auctions and bidding*, Journal of Economic Literature, 25 (1987), pp. 699–738.

[26] P. R. Milgrom and R. J. Weber, *A theory of auctions and competitive bidding*, Econometrica, 50 (1982), pp. 1089–1122.

[27] T. R. Palfrey, *Multiple-object, discriminatory auctions with bidding constraints: a game-theoretic analysis*, Management Science. Journal of the Institute of Management Science. Application and Theory Series, 26 (1980), pp. 935–945.

[28] D. C. Parkes and L. H. Ungar, *Iterative combinatorial auctions: Theory and practice*, in Proceedings of the 18th National Conference on Artificial Intelligence, 2000. To appear.

[29] S. J. Rassenti, V. L. Smith, and R. L. Bulfin, *A combinatorial mechanism for airport time slot allocation*, Bell Journal of Economics, 13 (1982), pp. 402–417.

[30] B. A. Reed, *Finding approximate separators and computing tree width quickly*, in Proceedings of the 24th Annual ACM Symposium on Theory of Computing, 1992, pp. 221–228.

[31] N. Robertson and P. D. Seymour, *Graph minors. II. Algorithmic aspects of tree-width*, Journal of Algorithms, 7 (1986), pp. 309–322.

[32] D. J. Rose, R. E. Tarjan, and G. S. Lueker, *Algorithmic aspects of vertex elimination on graphs*, SIAM Journal on Computing, 5 (1976), pp. 266–283.

[33] M. H. Rothkopf, A. Pekeč, and R. M. Harstad, *Computationally manageable combinatorial auctions*, Management Science, 44 (1998), pp. 1131–1147.

[34] T. Sandholm and S. Suri, *Improved algorithms for optimal winner determination in combinatorial auctions and generalizations*, in Proceedings of the 18th National Conference on Artificial Intelligence, 2000, pp. 90–97.

[35] R. Wilson, *Strategic analysis of auctions*, in Handbook of Game Theory with Economic Applications, R. J. Aumann and S. Hart, eds., vol. 1, Elsevier Science, New York, NY, 1992, pp. 227–279.