Information Systems Research
Vol. 16, No. 2, June 2005, pp. 169–185
ISSN 1047-7047 | EISSN 1526-5536 | 05 | 1602 | 0169

informs

DOI 10.1287/isre.1050.0052

Toward Comprehensive Real-Time Bidder Support in Iterative Combinatorial Auctions

Gediminas Adomavicius, Alok Gupta
Department of Information and Decision Sciences, Carlson School of Management, University of Minnesota, 321 19th Avenue South, Minneapolis, Minnesota 55455 {gedas@umn.edu, agupta@csom.umn.edu}

Many auctions involve selling several distinct items simultaneously, where bidders can bid on the whole or any part of the lot. Such auctions are referred to as combinatorial auctions. Examples of such auctions include truck delivery routes, industrial procurement, and FCC spectrum. Determining winners in such auctions is an NP-hard problem, and significant research is being conducted in this area. However, multiple-round (iterative) combinatorial auctions present significant challenges in bid formulations as well. Because the combinatorial dynamics in iterative auctions can make a given bid part of a winning and nonwinning set of bids without any changes in the bid, bidders are usually not able to evaluate whether they should revise their bid at a given point in time or not. Therefore, in this paper we address various computational problems that are relevant from the bidder's perspective. In particular, we introduce two bid evaluation metrics that can be used by bidders to determine whether any given bid can be a part of the winning allocation and explore their theoretical properties. Based on these metrics, we also develop efficient data structures and algorithms that provide comprehensive information about the current state of the auction at *any* time, which can help bidders in evaluating their bids and bidding strategies. Our approach uses exponential memory storage but provides fast incremental update for new bids, thereby facilitating bidder support for real-time iterative combinatorial auctions.

Key words: iterative combinatorial auctions; real-time bidder support; bid evaluation metrics; computational techniques for combinatorial auctions
History: Salvatore T. March, Senior Editor; Paulo Goes, Associate Editor. This paper was with the authors 5 months for 2 revisions.

1. Introduction

Online auctions are among the most widely used electronic market mechanisms on the Internet. In a forward auction, a seller sells one or several objects (goods) to several potential buyers. In typical single-object auctions, determining the auction's winner and its revenue is a computationally tractable problem. In simultaneous auctions of several distinct items, it is often desirable to allow bids on combinations of items because of complementarities or positive externalities (Hudson and Sandholm 2002). Such auctions are called *combinatorial auctions*. It has been shown that combinatorial auctions can lead to more efficient allocations than traditional auction mechanisms when buyers' valuations of the items are dependent on the specific combination of items they are able to win (Hudson and Sandholm 2002, de Vries and Vohra 2003). However, the winner determination problem in such auctions is computationally intractable, i.e., NP-hard (Rothkopf et al. 1998, Sandholm 1999). Therefore, researchers have focused primarily on the winner determination problem, and the previous work can be broadly categorized into: (1) development of heuristics for solving the general winner determination problem (Sandholm 1999, 2002) and (2) identification of tractable special cases of the winner determination problem (Rothkopf et al. 1998, Tennenholtz 2000).[1]

In multiround (or iterative) open auctions, bidders need to evaluate the efficacy of their bids. In single-unit iterative auctions (e.g., English auctions), if a bidder is not the highest bidder, she needs to bid an amount higher than the current highest bid to have a chance to win the auction. However, in combinatorial

[1] The complete coverage of extensive research in these areas is beyond the scope of this paper; de Vries and Vohra (2003) present an extensive up-to-date bibliography of research in combinatorial auctions.

auctions this is not necessarily the case. A bid that is not among the current winners[2] can be among the future winners based simply on the combinations of later bids. For example, if there are three items A, B, and C, and the current bids are: (i) \$10 for AB and (ii) \$5 for A, the second bid is not currently winning. However, if a new bid of \$6 for BC arrives, then, assuming the auctioneer is maximizing his revenue, bid (ii) will be among the winning bids. As the example demonstrates, it may not be in the best interest of a bidder to always update their bids. Unfortunately, due to the complexity of the winner determination problem, bidders in iterative combinatorial auctions usually do not know whether their bid is currently winning or whether it has the potential to become winning at some point in the future.

Prior research in iterative combinatorial auctions has primarily focused on creating specific mechanisms that either explore the intricacies of a specific application or create rules and restrictions to allow the creation of several well-defined rounds of bidding. An example of mechanisms for specific applications is the BICAP mechanism created by Brewer and Plott (1996) for the rights to use railroad tracks. Examples of creating specific rules to enable multiround computation of winner determination include iBundle by Parkes (1999), Ausubel and Milgrom (2002), and Rothkopf et al. (1998). Other interesting ideas include shifting the onus of bid evaluation to the bidders. For example, Banks et al. (1989) create a mechanism where it is the responsibility of the bidders to look at the existing bids and submit a new bid that makes the combined "package" optimal; Kelly and Steinberg (2000) describe a similar auction environment for use in assigning carrier of last resort (COLR) responsibility to U.S. telephone carriers that undertake various obligations as a condition for receipt of universal service support. Other researchers, such as Nisan (2000), provide bidding languages so that bidders can represent their preferences (bundle/bid combinations) in sealed bid (noniterative) auctions.

Pekec and Rothkopf (2003) note that the continuous-time combinatorial auctions are difficult to conduct for more than a handful of items. In their opinion, it is preferable to design mechanisms that identify discrete rounds with specific rules, making winner determination efficient. However, they note that if it is not possible to create discrete rounds, then "*bidtakers should take particular care in providing tools that help bidders in bid preparation*" (p. 1,501). Overall, it is fair to say that the issue of providing comprehensive real-time information to bidders *during* the auction has been largely untouched.

In this paper, we take a different approach and address various computational problems that are relevant from the bidder's perspective. In other words, we provide theoretical, algorithmic, and computational foundations of a *bidder support system* for iterative combinatorial auctions. Following the design science paradigm of Hevner et al. (2004), we create novel *design artifacts* by designing data structures and algorithms that provide comprehensive information about the current auction state to bidders in iterative combinatorial auctions. Auction state information is essential for bidders in formulating and evaluating their bids in any iterative auction; hence, our approach clearly satisfies the *problem relevance* criterion of design science research. We use rigorous analytical and computational testing approaches to perform *design evaluation* of our *design artifacts*, satisfying the formal *research rigor* criterion of design science research. Cumulatively, this research satisfies the *research contribution* dimension of design science research in multiple ways by contributing to the fundamental state of knowledge about properties of bid space and enabling the bidder support system for interactive combinatorial auctions. Specifically, we introduce two bid evaluation metrics that can be used by bidders to determine whether any given bid can be a part of the winning allocation and develop computationally fast algorithms that calculate these metrics. The introduction of these bid evaluation metrics leads to the following significant results that satisfy the *design as search process* criterion of design science research (Hevner et al. 2004): (a) It facilitates access to the crucial bid-related information to the bidders anytime during the course of an auction as opposed to traditional approaches that only provide final allocation information; (b) because our algorithms and data structures provide access to the complete state

[2] A current winner is defined as a set of bids that would win the allocation of desired items if the auction were to stop at that point in time.

of a combinatorial auction in real time, winners can be easily determined incrementally and are known after each new bid; (c) development of new bid evaluation metrics can offer insights into the underlying structure of combinatorial auctions and provide novel directions for future research.

The prior research indicates that exhaustive state space-based approach (or complete enumeration approach) for winner determination in combinatorial auction is only feasible for cases for less than a dozen items (Sandholm 2002, Pekec and Rothkopf 2003). Moreover, the computational performance of such algorithms is reported to be unsatisfactory for real-time use (de Vries and Vohra 2003). In this paper, we extend these boundaries to a much larger number (25–30). We provide theoretical foundations of the interrelationships and trade-offs among the exponential number of possible bids. Based on these relationships, we define some important constructs, such as subauctions as well as live, dead, and winning bids. We then develop data structures that support real-time bidder queries and an efficient algorithm for keeping these data structures up to date throughout the duration of an auction. We then use this algorithm to perform extensive computational experiments to demonstrate the performance of our implementation. The experimental results indicate that our implementation is capable of providing real-time bidder support for auction sizes where even a single-time winner determination problem is considered challenging.

2. Combinatorial Auction Dynamics: Theoretical Analysis

2.1. Preliminaries and Definitions

Let $\mathcal{I}$ be the set of items to be auctioned, and let $N = |\mathcal{I}|$. We use the terms *auction set* and *auction size* to refer to $\mathcal{I}$ and N, respectively. In a combinatorial auction, participants (person, software agent, etc.) can place bids on any *itemset*, i.e., any nonempty subset of $\mathcal{I}$.

An arbitrary bid b can be represented by the tuple $b = (S, v, t)$, where $S \subseteq \mathcal{I}$ $(S \neq \varnothing)$, $v \in \mathbb{R}^+$, and $t \in \mathbb{R}^+$. Here S denotes the itemset the bid was placed on, also called the *span* of the bid; v denotes the *value* of the bid (e.g., the monetary amount specified in this bid); and t denotes the *time* the bid is placed. Given bid b, we use the notation $S(b)$, $v(b)$, and $t(b)$ to refer to the span, value, and time of the bid, respectively. In this paper we assume that, once submitted, bids cannot be withdrawn from the auction.

We assume that there exists a strict chronological order to bids, i.e., that no two bids arrive at exactly the same moment in time, or more formally, for any two different bids b' and b'' we always have $t(b') \neq t(b'')$. Therefore, all bids in an auction can be ordered according to this order (i.e., $b_1, b_2, b_3, \ldots$). This is a natural assumption (commonly accepted in auction literature) and is used for tie-breaking purposes. For example, if we have two bids with identical spans and identical values, the earlier bid is typically preferred over the later one.

Because of the existence of the strict chronological order of bids, we can introduce the notion of the *auction states*, which are represented by nonnegative integers. In particular, auction state k (where $k = 0, 1, 2, \ldots$) refers to the auction after the first k bids are submitted. This bid set is denoted as B_k, i.e., $B_k = \{b_1, \ldots, b_k\}$. Auction state 0 refers to the auction before any bids are made, i.e., $B_0 = \varnothing$. Obviously, $B_k \subseteq B_l$, for any k and l such that $k \leq l$.

Given an arbitrary set of bids B in a combinatorial auction, a bid set C (where $C \subseteq B$) is called a *bid combination in* B if all bids in C have nonoverlapping spans, i.e., for every $b', b'' \in C$ where $b' \neq b''$, we have $S(b') \cap S(b'') = \varnothing$. The set of *all* bid combinations possible at auction state k is denoted as $\mathbb{C}_k$, i.e., $\mathbb{C}_k = \{C \subseteq B_k \mid b', b'' \in C,\ b' \neq b'' \Rightarrow S(b') \cap S(b'') = \varnothing\}$.

We assume that the winners of the auction are determined by maximizing the seller's revenue, i.e., $\max_{C \in \mathbb{C}_k} \sum_{b \in C} v(b)$. The bid combination that maximizes this expression is called a *winning bid combination* and is denoted as WIN_k (for auction state k). Moreover, given auction state k, bid $b \in B_k$ is called a *winning bid* in B_k if $b \in WIN_k$. Furthermore, if bid $b \in B_k$ is not a winning bid in B_k and cannot possibly be a winning bid in *any* subsequent auction state, then b is called a *dead bid* in B_k. Formally, bid $b \in B_k$ is dead if $(\forall B_l \supseteq B_k)$ $(b \notin WIN_l)$. The set of all dead bids in B_k is denoted as $DEAD_k$. On the other hand, if $b \notin DEAD_k$, then bid $b \in B_k$ is called a *live bid* in B_k. The set of all live bids in B_k is denoted as $LIVE_k$. Based on the definitions of WIN_k, $DEAD_k$, and $LIVE_k$, note that: $DEAD_k \cap LIVE_k = \varnothing$, $DEAD_k \cup LIVE_k = B_k$, $WIN_k \subseteq LIVE_k$, and $DEAD_k \subseteq DEAD_h$ for any $h \geq k$.

To determine the winning bid combination, we need to establish a preference order on various bid combinations. Note that because a bid combination is a set of bids with nonoverlapping spans, we can straightforwardly extend span, value, and time notions from bids to bid combinations as follows. Let C be a bid combination. Then $S(C)$, $v(C)$, and $t(C)$ are defined as follows:

$$S(C)=\bigcup_{b\in C} S(b), \quad v(C)=\sum_{b\in C} v(b), \quad \text{and} \quad t(C)=\max_{b\in C} t(b).$$

Also, for the purpose of notational completeness, we define the span, value, and time of an empty bid combination $\varnothing$ as follows: $S(\varnothing)=\varnothing$, $v(\varnothing)=0$, and $t(\varnothing)=0$.

To incorporate a strict tie-breaking mechanism into the winner determination process, we will define a *strict total order* for sets of bid combinations by defining a $\prec$ relation on bid combinations. In other words, given an arbitrary set of bid combinations $\mathbb{C}$, for any $C', C'' \in \mathbb{C}$ such that $C' \neq C''$, we will *always* have that either $C' \prec C''$ (in this case, we will say that C'' is *better* than C') or $C'' \prec C'$ (i.e., C' is better than C''). According to the revenue maximization requirement that combinatorial auctions typically follow, relation $\prec$ must have the following "value monotonicity" property: $(\forall C', C'' \in C_k)$ $(C' \prec C'' \Rightarrow \nu(C') \leq \nu(C''))$. In other words, C'' cannot be better than C' if C'' provides less value than C'. Therefore, the above property guarantees that the best bid combination will always be the one with the maximal revenue. However, this property alone is not sufficient for relation $\prec$ to be a strict total order, because a combinatorial auction can have several bid combinations with the (same) maximal revenue. Therefore, in addition to the above value monotonicity property, relation $\prec$ must have a tie-breaking mechanism to be able to order bid combinations with equal values. The following example illustrates the importance of the tie-breaking mechanism.

Example 1. Consider a five-item auction, i.e., $\mathcal{I} = \{a, b, c, d, e\}$. Furthermore, assume that the following five bids were made in the exact chronological sequence shown here:

$$b_1=(ab, 15), \qquad b_2=(bc, 10), \qquad b_3=(ad, 10),$$
$$b_4=(cd, 5), \qquad b_5=(e, 5).$$

The maximal possible value of a bid combination in the above example is 25, and we have two bid combinations, i.e., $\{b_1, b_4, b_5\}$ and $\{b_2, b_3, b_5\}$, that have such value. Which one should be picked as a winning combination? If each combination consisted of a single bid, then we could straightforwardly break the tie by choosing the earlier bid. However, bid combinations $\{b_1, b_4, b_5\}$ and $\{b_2, b_3, b_5\}$ are not straightforwardly comparable in terms of time. For example, excluding the "overlap" bid b_5, which is present in both bid combinations, combination $\{b_1, b_4\}$ started earlier than $\{b_2, b_3\}$ (because of b_1), but $\{b_2, b_3\}$ ended earlier than $\{b_1, b_4\}$ (because of b_4). □

While several different tie-breaking mechanisms for relation $\prec$ are possible, because of the space limitations we consider one specific tie-breaking approach in this paper.[3] In this approach, if there are several bid combinations with equal value, the earliest *completed* bid combination is chosen over those completed later (excluding overlap, as will be shown below). Justification for this is straightforward: We do not want to change the winning bid combination as long as the auction revenue does not increase. Formally, we will state that C'' is better than C' (i.e., $C' \prec C''$), if either of the following two conditions is true:

(i) $v(C') < v(C'')$; or
(ii) $v(C') = v(C'')$, and $t(C' \setminus C'') > t(C'' \setminus C')$.

Note that here we use $t(C' \setminus C'') > t(C'' \setminus C')$ instead of the simpler $t(C') > t(C'')$ condition in order to eliminate the "overlap" between the bid combinations, i.e., we eliminate bids that are present in both C' and C''. Based on the fact that $(C' \setminus C'') \cap (C'' \setminus C') = \varnothing$ and that there exists a strict chronological order of individual bids, we always have that $t(C' \setminus C'') \neq t(C'' \setminus C')$. This makes condition (ii) a true tie-breaker, because it is not possible to have $t(C' \setminus C'') = t(C'' \setminus C')$ when $C' \neq C''$. Furthermore, it is straightforward to show formally that $\prec$ is truly a strict total order, i.e., that it is irreflexive (i.e., $C \prec C$ is not true for any C), transitive ($C' \prec C''$ and $C'' \prec C'''$ implies $C' \prec C'''$), and total (as was demonstrated earlier in this paragraph).

Example 2. Using the above definition of strict total order on bid combinations, the winning bid combination in Example 1 would be $\{b_2, b_3, b_5\}$. In other

[3] Most of the theoretical results presented in this paper also hold for other strict total order relations $\prec$ that have the value monotonicity property.

words, we have that $\{b_1, b_4, b_5\} \prec \{b_2, b_3, b_5\}$, because $v(\{b_1, b_4, b_5\}) = v(\{b_2, b_3, b_5\}) = 25$, but

$$t(\{b_1, b_4, b_5\}\backslash\{b_2, b_3, b_5\}) = t(\{b_1, b_4\}) = t(b_4) > t(b_3) = t(\{b_2, b_3\}) = t(\{b_2, b_3, b_5\}\backslash\{b_1, b_4, b_5\}). \quad \square$$

Once the strict total order is defined on bid combinations, we can rewrite the winner determination problem simply as: $WIN_k = \max_{\prec} \mathbb{C}_k$. With the strict total order on bid combinations in place, we can now derive a multitude of results about various aspects of combinatorial auctions.

2.2. Subauctions and Their Winning Bid Combinations

Because we want to be able to provide a bidder with the information regarding the auction at every instant, we need to know the currently winning bids at each stage of the auction. For representational simplicity we do this by defining the concept of subauction: Given auction state k and itemset X, define $B_k[X]$ as $B_k[X] = \{b \in B_k \mid S(b) \subseteq X\}$. We can then say that each itemset X ($X \subseteq \mathcal{I}$) defines a *subauction* of the overall auction $\mathcal{I}$.

Furthermore, given auction state k and itemset X, $\mathbb{C}_k[X]$ denotes the set of all bid combinations in $B_k[X]$, i.e., $\mathbb{C}_k[X] = \{C \subseteq B_k[X] \mid b', b'' \in C,\ b' \neq b'' \Rightarrow S(b') \cap S(b'') = \varnothing\}$. Note that, by definition, $B_k \equiv B_k[\mathcal{I}]$ and $\mathbb{C}_k \equiv \mathbb{C}_k[\mathcal{I}]$. The following table states three intuitive properties of $B_k[X]$ and $\mathbb{C}_k[X]$ that follow immediately from their definitions. In other words, for any nonempty itemsets X, Y, and any auction state k, all the statements in Table 1 are true.

Given the strict total order on bid combinations, the winner determination problem for each subauction is formulated as: $WIN_k[X] = \max_{\prec} \mathbb{C}_k[X]$, where $WIN_k[X]$ represents the winning bid combination for subauction X at auction state k. Note that, by definition, $WIN_k \equiv WIN_k[\mathcal{I}]$ at each stage k ($k = 1, 2, \ldots$). Again, for the purpose of notational completeness we define $WIN_k[\varnothing] = \varnothing$.

Table 1 **Various Properties of $B_k[X]$ and $\mathbb{C}_k[X]$**

Description	Properties of $B_k[X]$	Properties of $\mathbb{C}_k[X]$
Monotonicity w.r.t. auction state	$B_k[X] \subseteq B_{k+1}[X]$	$\mathbb{C}_k[X] \subseteq \mathbb{C}_{k+1}[X]$
Monotonicity w.r.t. subauction	$(X \subseteq Y) \Rightarrow (B_k[X] \subseteq B_k[Y])$	$(X \subseteq Y) \Rightarrow (\mathbb{C}_k[X] \subseteq \mathbb{C}_k[Y])$
"Superadditivity" w.r.t. subauction	$B_k[X \cup Y] \supseteq B_k[X] \cup B_k[Y]$	$\mathbb{C}_k[X \cup Y] \supseteq \mathbb{C}_k[X] \cup \mathbb{C}_k[Y]$

To be able to keep track of winning bids at each state of the auction, one of the important issues is to understand the iterative dynamics of winning bid combinations, i.e., how the current winning bid combination changes when the new bid is submitted to the auction. Suppose that we are at auction state k (i.e., k bids have been submitted so far) and a new bid b_{k+1} is placed. The following two theorems explain the dynamics of the winning bid combination for each subauction X, i.e., the relationship between $WIN_k[X]$ and $WIN_{k+1}[X]$ for each subauction X.

THEOREM 1. $(\forall X \not\supseteq S(b_{k+1}))\ (WIN_{k+1}[X] = WIN_k[X])$.[4]

THEOREM 2. $(\forall X \supseteq S(b_{k+1}))(WIN_{k+1}[X] = \max_{\prec}[\{b_{k+1}\} \cup WIN_k[X\backslash S(b_{k+1})], WIN_k[X]])$.

Simply put, Theorem 1 says that for all subauctions X that b_{k+1} does not belong to (i.e., $S(b_{k+1}) \not\subseteq X$ or, in other words, $b_{k+1} \notin B_{k+1}[X]$), the winning bid combination does not change. Theorem 2 indicates that for all subauctions X that b_{k+1} does belong to, the winning bid combination of subauction X can change only if the combination of new bid b_{k+1} and the winning bids in the $X\backslash S(b_{k+1})$ subauction is better than the previous winning bid combination of subauction X. Note that, for the special case where $X = S(b_{k+1})$, Theorem 2 is simplified to $WIN_{k+1}[X] = \max_{\prec}[WIN_k[X], \{b_{k+1}\}]$, because $WIN_k[\varnothing] = \varnothing$.

Let $VL_k(X)$ be the *value level* of subauction X at auction state k or, more formally, $VL_k(X) = v(WIN_k[X])$. In other words, $VL_k(X)$ represents the maximal possible revenue from subauction X at auction state k. Note that the revenue of the entire auction is then denoted as $VL_k(\mathcal{I})$. Based on the definition of $VL_k(X)$, we can derive various properties of $VL_k(X)$, as stated in the following lemma.

LEMMA 1. *For any auction state k and itemsets X, Y, the following statements are true*:

1. $VL_k(X) \leq VL_{k+1}(X)$;
2. $(X \subseteq Y) \Rightarrow (VL_k(X) \leq VL_k(Y))$;
3. $(X \cap Y = \varnothing) \Rightarrow (VL_k(X) + VL_k(Y) \leq VL_k(X \cup Y))$.

[4] Proofs of all theoretical results are provided in the appendix.

In other words, $VL_k(X)$ (or the revenue of subauction X at auction state k) is monotonically nondecreasing with respect to both auction state k and subauction X. It is also superadditive with respect to the subauction—the cumulative revenue from two nonoverlapping subauctions cannot be greater than the revenue of the combined subauction.

2.3. "Deadness" and "Winning" Levels of Bids and Their Theoretical Properties

In this section we derive several theoretical results about dead, live, and winning bids, including the necessary and sufficient conditions for determining whether a given bid is dead, live, or winning at each state of a combinatorial auction. We begin with the following theorem, which identifies an important correspondence between the live bids of the auction and the winning combinations of individual subauctions.

THEOREM 3. *Given auction state k and bid $b \in B_k$, such that $S(b) = X$, we have*: $b \in LIVE_k \Leftrightarrow b \in WIN_k[X]$.

The above theorem states that bid b is live at auction state k if and only if b is a winning bid in a subauction defined by its span $S(b)$ at auction state k. Furthermore, because $b \in WIN_k[X]$ and $X = S(b)$, the winning combination of subauction X consists of bid b alone, i.e., $b \in LIVE_k \Leftrightarrow WIN_k[S(b)] = \{b\}$. Therefore, as the next corollary indicates, the set of live bids in a combinatorial auction is comprised of the winning bids from all subauctions.

COROLLARY 3A. $LIVE_k = \bigcup_{X \subseteq \mathcal{I}} WIN_k[X]$.

In other words, at any auction state, the set of all live bids in an auction is the same as the set of winning bids from all subauctions. Furthermore, from the bidder's perspective it would be useful to know how much she should bid on an itemset to guarantee that her bid is live. The next theoretical result addresses this issue, i.e., given auction state k and a newly submitted bid b_{k+1}, the following corollary (based on Theorem 3) states the necessary and sufficient condition for the deadness of b_{k+1}.

COROLLARY 3B. *Given auction state k and new bid b_{k+1}, such that $S(b_{k+1}) = X$, the following is true*: $b_{k+1} \in DEAD_{k+1} \Leftrightarrow v(b_{k+1}) \leq VL_k(X)$.

Simply put, any new bid on itemset X will be dead if and only if the current winning combination of subauction X has the same or greater value than the value of the new bid. Therefore, while $VL_k(X)$ was introduced to denote the value of subauction X at auction state k, it also represents the "deadness level" for any new bids on itemset X at auction state k. Thus, we will also call $VL_k(X)$ the *deadness level* of itemset X at auction state k and will denote it as $DL_k(X)$. More precisely, for any auction state k and itemset X we have: $DL_k(X) = VL_k(X)$. Note that the above corollary can be straightforwardly formulated as the necessary and sufficient condition for "liveness" of the new bid b_{k+1}, i.e., $b_{k+1} \in LIVE_{k+1} \Leftrightarrow v(b_{k+1}) > VL_k(X)$. Note that while in this paper we provide the precise (necessary and sufficient) condition for bid deadness, weakened sufficient conditions have been noted in prior literature. For example, Rothkopf et al. (1998) remove any bid that is not the highest bid on its span.

Next, Theorem 4 defines the necessary and sufficient condition for a bid to be a winning bid.

THEOREM 4. *Given auction state k and new bid b_{k+1}, such that $S(b_{k+1}) = X$: $b_{k+1} \in WIN_{k+1} \Leftrightarrow v(b_{k+1}) > VL_k(\mathcal{I}) - VL_k(\mathcal{I} \setminus X)$.*

Let us denote $WL_k(X)$ to be a *winning level* for bids on itemset X at auction state k and define it as: $WL_k(X) = VL_k(\mathcal{I}) - VL_k(\mathcal{I} \setminus X)$. Such a definition is appropriate because (based on Theorem 4) $b_{k+1} \in WIN_{k+1}$ if and only if $v(b_{k+1}) > WL_k(X)$. Also note that because of our definition of $WIN_k[\varnothing] = \varnothing$, Theorem 4 provides the correct result for the special case where the new bid b_{k+1} is placed on the whole auction set $\mathcal{I}$, i.e., when $S(b_{k+1}) = \mathcal{I}$. Obviously, the winning level for itemset $\mathcal{I}$ should be equal to the current auction revenue $VL_k(\mathcal{I})$, and from Theorem 4 we obtain exactly that: $WL_k(\mathcal{I}) = VL_k(\mathcal{I}) - VL_k(\mathcal{I} \setminus \mathcal{I}) = VL_k(\mathcal{I}) - VL_k(\varnothing) = VL_k(\mathcal{I})$ because $VL_k(\varnothing) = v(WIN_k[\varnothing]) = v(\varnothing) = 0$.

Numerous properties of $DL_k(X)$ and $WL_k(X)$ can be derived from their definitions and earlier theoretical results. Some interesting properties are stated in the following theorem.

THEOREM 5. *For any auction state k and itemsets X, Y, the following statements are true*:

1. $DL_k(X) \leq WL_k(X)$;
2. $DL_k(\mathcal{I}) = WL_k(\mathcal{I})$;
3. $WL_k(X) = DL_k(\mathcal{I}) - DL_k(\mathcal{I} \setminus X)$;

4. $DL_k(X) = WL_k(\mathcal{I}) - WL_k(\mathcal{I} \backslash X)$;
5. $DL_k(X) \leq DL_{k+1}(X)$;
6. $X \subseteq Y \Rightarrow DL_k(X) \leq DL_k(Y)$;
7. $X \subseteq Y \Rightarrow WL_k(X) \leq WL_k(Y)$;
8. $X \cap Y = \varnothing \Rightarrow DL_k(X \cup Y) \geq DL_k(X) + DL_k(Y)$;
9. $b \in LIVE_k$, s.t. $S(b) = X \Rightarrow DL_k(X) = v(b)$;
10. $b \in WIN_k$, s.t. $S(b) = X \Rightarrow WL_k(X) = v(b)$.

The above results can provide some interesting insights about combinatorial auctions and bid dynamics. Statement (1) is very intuitive: When bidding on itemset X at auction state k, the winning level for X can never be lower than the deadness level of X. However, in some special cases the deadness and winning levels can be equal, e.g., it is always the case for subauction $\mathcal{I}$, as statement (2) indicates. In addition, based on Statements (9) and (10), we have that $DL_k(X) = WL_k(X)$ for all subauctions X that represent a span of a currently winning bid, i.e., all subauctions X, such that $\exists b \in WIN_k$ where $S(b) = X$. This is the case, because $b \in WIN_k \Rightarrow b \in LIVE_k$ and, by combining (9) and (10), we have that $WL_k(X) = v(b) = DL_k(X)$.

Furthermore, Statements (3) and (4) suggest a certain symmetry (or duality) between deadness and winning levels. Moreover, Statements (6) and (7) show that both deadness and winning levels are monotonically nondecreasing with respect to the itemset. In addition, the deadness levels are monotonically nondecreasing with respect to the auction state, as presented in Statement (5); it is easy to show that winning levels do not have the same property. For example, assume that we have a two-item combinatorial auction, i.e., $|\mathcal{I}| = \{a, b\}$, where the first bid was $(ab, 10)$ and the second bid was $(b, 5)$. According to the derived theoretical results, $WL_k(a) = VL_k(\mathcal{I}) - VL_k(\mathcal{I} \backslash a) = VL_k(ab) - VL_k(b)$. Therefore, $WL_1(a) = 10 - 0 = 10$ and $WL_2(a) = 10 - 5 = 5$, and consequently, $WL_1(a) > WL_2(a)$.

Finally, while Statement (8) in Theorem 5 does indicate that the deadness levels are superadditive with respect to the itemset, in general the same is not true for winning levels. This can be illustrated using the same auction example as in the previous paragraph, where we have that $WL_2(a) = 5$, $WL_2(b) = 10$, and $WL_2(ab) = 10$. Consequently, $WL_2(ab) < WL_2(a) + WL_2(b)$.

The following lemma provides some intuition about the inherent complexity of combinatorial auctions by demonstrating that the number of live bids at a given state of an auction can possibly be exponential with respect to auction size N.

Lemma 2. *In a combinatorial auction of size N:*

1. The number of live bids at any auction state cannot be greater than $2^N - 1$;
2. $2^N - 1$ is a tight upper bound, i.e., it is possible to have an auction state where there are exactly $2^N - 1$ live bids.

However, it is easy to see that, while the number of live bids can be exponential, the number of winning bids can never be greater than N. This is the case, because any bid combination (including the winning bid combination WIN_k) contains only bids with nonoverlapping spans, i.e., $S(WIN_k) \subseteq \mathcal{I} \Rightarrow |WIN_k| \leq |\mathcal{I}| = N$.

3. Implementing Bidder Support: Data Structures and Algorithms

3.1. Infrastructure for Bidder Support

Based on the theoretical results presented earlier in this paper, we have developed data structures that are able to store important auction state information (e.g., current deadness and winning levels for any itemset X, current winning bid combination) and provide bidders with efficient (real-time) access to it. Specifically, throughout the duration of the auction, we propose to use the following two arrays, each containing some information about every itemset X:

- `ValueLevel`, where `ValueLevel[X]` always (i.e., at any auction state k) contains the up-to-date $VL_k(X)$ for itemset X.
- `LastWinBid`, where `LastWinBid[X]` always (i.e., at any auction state k) contains the itemset of the last bid (in a chronological sense) that belongs to the current $WIN_k[X]$ for the subauction X.

The space complexity of these data structures is $O(2^N)$, because there are $2^N - 1$ possible nonempty itemsets in an auction of size N. As discussed in §2.3, it is possible to have up to $2^N - 1$ live bids in such an auction (Lemma 2), where the value of each bid represents a deadness level for a specific itemset (Theorem 5, Statement 9). Therefore, intuitively, such expo-

nential data structures are necessary if we want to guarantee a very fast (i.e., constant-time) access to crucial bid information, such as itemset deadness levels, that will be discussed in more detail in §3.2. Therefore, for practical purposes we have that N is bounded by 25–30 or so, because the array access is most efficient when the whole array fits into the main memory and no OS swapping/paging needs to be performed. However, as discussed in §1, it is still a large improvement over the previously known complete enumeration approaches.

There are several challenges related to efficient implementation of bidder support and, more specifically, to the above data structures:

- How can data be represented (or more specifically, itemsets) efficiently in the above data structures? This issue is discussed in the remainder of this section.
- What auction and bid-related information can be extracted (queried) by bidders from these data structures in real time? This issue is addressed in §3.2.
- How can these (exponential) data structures be kept up to date after every single bid in real time? We will address this issue in §3.3.

We use *bitmaps* to represent itemsets in both of the above arrays. Bitmaps are commonly used to represent setlike data in many different applications, mainly because they allow for a concise representation of sets and provide for an efficient (constant-time) implementation of many standard set operations (set union, intersection, difference, etc.). For these reasons, our implementation uses a bitmap-based data structure to represent the sets of items specified in the bids placed by the auction participants.

More specifically, assume $\mathcal{I}$ is the set of items that is being auctioned, where $|\mathcal{I}| = N$, and each item is encoded as a number between 0 and $N-1$, i.e., $\mathcal{I} = \{0, 1, \ldots, N-1\}$. Then, any possible itemset X (i.e., $X \subseteq \mathcal{I}$) can be represented by a sequence of bits (i.e., a bitmap) of length N, where ith bit ($i = 0, 1, \ldots, N-1$) in a bitmap is set to 1 if $i \in X$; otherwise, it is set to 0. There exists a straightforward one-to-one correspondence between the set of all subsets of $\mathcal{I}$, the set of all bitmaps of length N, and the set of integers $\{0, 1, \ldots, 2^N - 1\}$ because there are 2^N different bitmaps of length N, and each of them constitutes a binary representation of some integer between 0 and $2^N - 1$. In other words, an arbitrary itemset X corresponds to integer $\sum_{i \in X} 2^i$. Therefore, a standard 4-byte (32-bit) integer can represent any subset of $\mathcal{I}$, where $|\mathcal{I}| \leq 32$. Table 2 provides some examples of itemsets and their corresponding bitmaps (and integers) for the case $\mathcal{I} = \{0, 1, 2, 3\}$.

Table 2 Illustration of One-to-One Correspondence Between Itemsets, Bitmaps, and Integers

Itemset	Bitmap*	Integer
$\varnothing$	0000	0
{0}	0001	1
{1, 2}	0110	6
{0, 2, 3}	1101	13
{0, 1, 2, 3}	1111	15

*Here we use the traditional bitmap representation—the order of bits is right to left, i.e., 0th bit is the rightmost one.

As mentioned above, using bitmap representations of sets provides for an efficient implementation of many set operations. In fact, many set operations can be performed in constant time on bitmaps/integers using bitwise *NOT*, *AND*, *OR*, and *XOR* operations that are supported by most microprocessors and, consequently, by many programming languages (e.g., C, C++). Let $\widehat{X}$ denote a bitmap/integer representation of itemset X. Then, Table 3 illustrates how basic set operations can be implemented using bitmaps and logical bitwise operations.

3.2. Effective Infrastructure Querying: Obtaining Auction Information in Real Time

Assuming our data structures contain up-to-date information (we will discuss how to update them effectively in §3.3), below are examples of some scenarios that may be of interest to bidders (as well as

Table 3 Implementing Basic Set Operations Using Bitmaps

Set operation	Set notation	Bitwise operation
Intersection	$X \cap Y$	$\widehat{X}$ and $\widehat{Y}$
Union	$X \cup Y$	$\widehat{X}$ or $\widehat{Y}$
Difference	$X \setminus Y$ (assuming $X \supseteq Y$)	$\widehat{X}$ xor $\widehat{Y}$
Symmetric difference	$X \div Y$ (i.e., $(X \setminus Y) \cup (Y \setminus X)$)	$\widehat{X}$ xor $\widehat{Y}$
Complement	$\overline{X}$ (i.e., $\mathcal{I} \setminus X$)	$\widehat{X}$ xor $\widehat{\mathcal{I}}$
Membership test	$a \in X$?	($\widehat{a}$ and $\widehat{X}$) $\neq 0$?
Subset test	$X \subseteq Y$?	($\widehat{X}$ and $\widehat{Y}$) $= \widehat{X}$?
Empty set test	$X = \varnothing$?	$\widehat{X} = 0$?

auctioneers) and their corresponding queries/calculations that are derived directly from the theoretical results presented earlier in this paper. Note that in the examples below we use the notation of itemsets and set operations (instead of bitmaps and bitwise logical operations) for better clarity.

Example 3 (Obtaining the Current Winning Level for Some Itemset). For example, "How much should I bid on itemset X for my bid to be a winning bid?"

Query: ValueLevel[$\mathcal{I}$] − ValueLevel[$\mathcal{I}\backslash$X].

This query is derived directly from the definition of the winning level and Theorem 4. □

Example 4 (Obtaining the Current Deadness Level for Some Itemset). For example, "How much should I bid on itemset X for my bid not to be a dead bid?"

Query: ValueLevel[X].

This query is derived directly from the definition of the deadness level and Corollary 3b. □

Example 5 (Obtaining the Current Revenue of the Auction). That is, "What would the total revenue be if the auction ended right now?"

Query: ValueLevel[$\mathcal{I}$].

This query is derived directly from the definition of the auction revenue. □

Assuming the auction data in the ValueLevel array is up to date, the computational complexity of the above three queries is $O(1)$ (in practice, no more than several microseconds), because they use only simple array lookups.

Example 6 (Obtaining the Winning Bid Combination). That is, "Which itemsets comprise the current winning bid combination?" The following query/procedure prints the spans (itemsets) and values of the bids from the currently winning bid combination:

```
(1) Curr = ℐ
(2) While LastWinBid[Curr] ≠ ∅ Do
(3)   Print LastWinBid[Curr],
        ValueLevel[LastWinBid[Curr]]
(4)   Curr = Curr\LastWinBid[Curr]
(5) End While
```

The above algorithm is derived from Theorem 2, i.e., based on the fact that a winning combination of any subauction X can be "decomposed" into the latest winning bid b and the winning bids of the "complement" auction $X\backslash S(b)$. The computational complexity of this algorithm is $O(|WIN_k|)$, which is minimal, because the same computational complexity is needed just to print the winning bid combination. In the worst case, this complexity is $O(N)$, i.e., linear in the number of items in the auction, because $|WIN_k| \le N$, as discussed earlier. Note that because N is usually less than 30 or so because of the real-life space complexity restrictions mentioned earlier, the winning bid allocation determination is also extremely efficient.

Also note that the above algorithm can be straightforwardly adapted to find the winning bid combination for any subauction X (not only for the entire auction set $\mathcal{I}$), in case such information is requested. For this purpose, simply replace $\mathcal{I}$ with X on Line 1.

3.3. Incremental Infrastructure Update

The incremental update of data structures is arguably the most crucial part of the proposed infrastructure—if we can update the auction state information effectively, then bidder-based queries can be performed in constant or near-constant time. Assume that we have the state k of the auction, and all data structures contain up-to-date values, i.e., ValueLevel[X] = $VL_k(X)$ and LastWinBid[X] contains the itemset of the last winning bid for subauction X. Let b_{k+1} be a new bid that is submitted to the auction. Furthermore, let S and v be the span and value of bid b_{k+1}, i.e., $S = S(b_{k+1})$ and $v = v(b_{k+1})$. We will assume that $b_{k+1} \in LIVE_{k+1}$, because, as demonstrated in §3.2 (Example 4), it takes one array lookup to retrieve the deadness level for any itemset, i.e., it only takes a constant time to determine whether an incoming bid is dead, in which case no update is needed and the bid can simply be discarded. Then, the data structures are updated for auction state $(k+1)$ as follows:

Input: new live bid b_{k+1} represented by its span S and value v.

```
(1) For all X ⊇ S
(2)   CandidateValue = v + ValueLevel[X\S]
(3)   If CandidateValue > ValueLevel[X] Then
(4)     ValueLevel[X] = CandidateValue
(5)     LastWinBid[X] = S
(6)   End If
(7) End For
```

This algorithm draws directly from the theoretical results presented earlier in this section (mainly from Theorems 1 and 2). More specifically, based on Theorem 1, new bid b_{k+1} does not affect the winning combination of any subauction X, such that $X \not\supseteq S(b_{k+1})$. Therefore, as Line 1 indicates, only subauctions $X \supseteq S(b_{k+1})$ have to be considered. Moreover, based on Theorem 2, new bid b_{k+1} can affect the winning combination of subauction $X \supseteq S(b_{k+1})$ only if $WIN_k[X] \prec \{b_{k+1}\} \cup WIN_k[X \backslash S(b_{k+1})]$. Consequently, based on the definition of $\prec$ and also on the fact that bid combination $WIN_k[X]$ precedes $\{b_{k+1}\} \cup WIN_k[X \backslash S(b_{k+1})]$ chronologically, the current winning combination of subauction X will change (i.e., $WIN_{k+1}[X] \neq WIN_k[X]$) only if $VL_k(X) < v(b_{k+1}) + VL_k(X \backslash S(b_{k+1}))$. Lines 2–3 in the above algorithm perform this check, and, if required, both of the data structures are updated on Lines 4–5. Specifically, Line 4 updates `ValueLevel[X]` with the new revenue of subauction X, i.e., the value of the new winning combination $\{b_{k+1}\} \cup WIN_k[X \backslash S(b_{k+1})]$. Also, Line 5 updates `LastWinBid[X]` with the span of the last winning bid, i.e., $S(b_{k+1})$.

The computational complexity of the above algorithm is $O(2^{N-|S|})$, because $2^{N-|S|}$ is the number of all possible supersets of S and only a constant amount of work is needed for each superset. To loop through all supersets X efficiently (in the `For` loop above), we use loopless Grey binary code generation approach (Bitner et al. 1976), which takes only a constant amount of time to calculate the next superset during each iteration. We present the experimental performance results of this algorithm in the next section.

4. Experimental Results

As mentioned earlier, problem spaces with more than a dozen items are considered too complex for the winner determination problem using complete enumeration approaches, and the computational performance of such algorithms is reported to be unsatisfactory for real-time use (de Vries and Vohra 2003). However, our implementation of the complete enumeration-based approach, presented in §§3 and 4, is able to handle much larger auction sizes (25–30) on a relatively obsolete machine.[5]

[5] All experiments were performed on a 450 MHz Pentium III computer with 256 MB RAM running Linux OS.

In this section we present some performance results of our algorithm for the incremental update of the comprehensive auction information (stored in `ValueLevel` and `LastWinBid` arrays) after each bid in real time. As discussed in §3.2, this information can be used to instantly determine current winners as well as various other bid-related characteristics (deadness, winning levels) at any time during the auction.

Our primary interest is in exploring the real-time capabilities of our approach, and, therefore, we performed a "throughput" test for our incremental update algorithm. In other words, given an auction set and a large number of incoming bids, we measure the time it takes for our incremental update algorithm to have the `ValueLevel` and `LastWinBid` data structures completely up to date after each bid. Because we were not aware of any publicly available large-scale (i.e., with tens of thousands or more bids) real-life bidding data sets, we have generated such data sets ourselves using (a) our own ad hoc random bid generator and (b) the Combinatorial Auction Test Suite (CATS) by Leyton-Brown et al. (2000), which has been used previously in combinatorial auctions literature for testing and benchmarking winner determination algorithms.

Table 4 presents the "throughput" test for our incremental update algorithm for the set of 500,000 randomly generated bids using our own ad hoc bid generator. The bids were generated using several simple schemes:

- [*Random span, random value*] For each bid in a data set, both the bid span and bid value were generated independently, i.e., both entities were picked uniformly at random from a fixed range of values

Table 4 Total Incremental Update Time to Handle 500,000 Bids (in Seconds)

	Bid generation scheme			
Auction size	Random span and random value	Random span and proportional value	Dynamic span (with $p = 1/2$) and proportional value	Dynamic span (with $p = 1/3$) and proportional value
24	379.6535	893.1401	210.2533	196.7459
20	23.4679	74.1205	11.6509	11.3079
16	1.0362	1.6946	1.0762	1.0368
12	0.5669	0.5763	0.5668	0.5660
8	0.5448	0.5472	0.5508	0.5205
4	0.5443	0.5461	0.5501	0.5682

independently of each other and of the previous bids in a data set;

• [*Random span, proportional value*] For each bid in a data set, the bid span was generated at random; however, the bid value was proportional to the size of the span, i.e., larger bid values were generated for larger itemsets;

• [*Dynamic span* ($p = 1/2$ *and* $p = 1/3$)*, proportional value*] For each bid in a data set, the bid span was generated dynamically by choosing an item at random and adding further items to it with "stopping" probability p (we considered cases where $p = 1/2$ and $p = 1/3$). In other words, in the $p = 1/2$ case, on average, 50% of the bids have single-item spans (i.e., we stopped after one item), 25% of the bids have two-item spans, 12.5% of the bids three-item spans, etc. Moreover, the bid value was generated to be proportional to the bid span. Clearly, the bid spans created in the $p = 1/3$ case tended to be larger than in case $p = 1/2$.

As Table 4 shows, even in the worst case our auction algorithm needed less than 15 minutes to handle 500,000 bids (or less than 1.79 milliseconds per bid, on average), while at the same time having to fully update all data structures after every one of the 500,000 bids that happened to be live at the time.

Note that because in the above experiment we generated bids in an ad hoc manner (as described earlier), only a small portion of the 500,000 bids are live upon being placed. It is one of the reasons for such effective performance, because, as discussed earlier, it only takes a constant time (i.e., one array lookup) to make sure whether an incoming bid is dead, in which case no update is needed and the bid is simply discarded. In our experiments, this deadness check was always performed in < 0.01 milliseconds (i.e., 0.003–0.006 ms for 24-item auctions, faster for smaller auctions). Therefore, having the comprehensive and up-to-date auction information (such as deadness levels) readily available at every state of the auction can actually facilitate better real-time performance.

One possible criticism of the above experiment could be that the bidding test sets are generated in an ad hoc manner and they are not representative of real-life auction dynamics. Therefore, we have also performed a similar "throughput" experiment using the bid sets generated by CATS 2.0 software. CATS software uses a suite of distribution families to generate realistic, economically motivated combinatorial bids that are consistent with some real-world domains (Leyton-Brown et al. 2000). For example, the *paths* distribution can be used to model bidding behavior for auctions involving truck routes, natural gas pipeline networks, network bandwidth allocation, or the use of railway tracks. Similarly, the *regions* distribution can be used to model bidding behavior for auctions involving real estate, drilling rights, or radio spectrum. Given a combinatorial auction of size $N = 24$, we used CATS software to generate 30 bidding scenarios of approximately 2,000 bids each for *every* available distribution (using default distribution parameters). Thus, we had about $\approx 60{,}000$ bids for each distribution. As expected, CATS software generated a much smaller portion of dead bids as compared to our ad hoc bid-generating scheme. However, the average incremental update time per bid was still only between 19–47 milliseconds. Even in the "perfect information" scenario, i.e., when we looked only at live bids, the average incremental update time per bid was between 107–644 milliseconds. Table 5 summarizes the results from this experiment.

Table 6 describes the relationship between the auction size, the span size of an incoming bid, and the time it takes to incrementally update the data structures after the new bid is placed. Each number presented in the table is an average of incremental update time for 100 live bids with a given span. Here we used only live bids, because, as discussed earlier, dead bids do not incur any updates. As reflected in Table 6, even for the largest auction the incremental

Table 5 **Incremental Update for 60,000 "Realistic" Bids Generated Using CATS Software for Combinatorial Auctions of Size $N = 24$**

Distribution	Total number of bids	Time (milliseconds)	
		Per bid	Per live bid
arbitrary	60,051	33.295	107.494
arbitrary-npv	60,078	29.079	113.509
arbitrary-upv	60,055	35.195	107.648
matching	60,087	47.195	378.059
paths	60,026	30.349	644.410
regions	60,059	20.015	187.477
regions-npv	60,077	19.296	199.354
regions-upv	60,080	20.904	169.926
scheduling	60,268	43.029	231.296

Table 6 Average Incremental Update Time for "Realistic" Live Bids Generated Using CATS Software (in Milliseconds)

Bid span size (s)	Auction size (N)								
	24	23	22	21	20	18	16	14	12
1	832.812	421.530	220.249	104.486	52.083	13.456	2.748	0.573	0.143
2	449.981	216.426	112.435	58.350	28.511	7.281	1.443	0.304	0.076
3	257.651	126.297	58.559	31.412	15.549	4.131	0.800	0.167	0.043
4	128.038	66.430	32.859	17.147	8.973	2.143	0.413	0.088	0.022
5	71.041	35.719	18.521	9.017	4.845	1.151	0.213	0.045	0.012
6	36.905	19.754	9.319	5.117	2.570	0.591	0.111	0.025	***
7	20.149	10.497	5.224	2.541	1.305	0.313	0.061	0.015	***
8	11.137	5.430	2.825	1.472	0.709	0.161	0.031	***	***

*** Less than 10 microseconds.

update times in worst cases (i.e., for single-item bids[6]) were under 1 second.

Furthermore, let $T(N, s)$ be the time needed to update the state of the auction (of size N) after a new live bid (with span of size s) arrives. Then, based on the empirical results in Table 6, note that $T(N, s) \approx 2 \cdot T(N, s+1)$ and $T(N, s) \approx 2 \cdot T(N-1, s)$. This is consistent with the theoretical computational complexity estimation discussed in §3.3, i.e., $T(N, s) = O(2^{N-s})$. This implies that increasing a bid span by one item would reduce the average incremental update time approximately by a factor of 2. Therefore, restricting the bid-span size (from below) is one of the possible heuristics that the auctioneer could use toward the end of the auction (when the bidding activity is typically the most intense) to increase the throughput of the proposed infrastructure, if needed. Obviously, reducing the auction size by one item would also reduce the average incremental update time approximately by a factor of 2. Also note that the proposed incremental update algorithm is straightforwardly parallelizable using the n-processor shared-memory model (i.e., by allocating the $(1/n)$th share of the affected subauctions to each of the n processors), which would allow further speeding up of the update of subauction information after each bid. While such parallelization would increase real-time capabilities of the auction, the ability to handle more items would not be affected because the storage requirements would stay the same (i.e., each processor must have access to entire arrays).

To illustrate the fundamental differences and comparative strengths/weaknesses of our proposed approach, we compare it with the optimal winner determination approach by Sandholm (2002). However, as mentioned earlier, it should be kept in mind that most prior research, including Sandholm (2002), has focused mainly on *one-time* winner determination algorithms (i.e., determining the winning bids after the auction is closed) that were not designed to provide any additional capabilities, such as providing winning and deadness levels for any possible bid in real time at any state of the auction.

The heuristic algorithm proposed by Sandholm (2002) uses a tree data structure for storing all submitted bids and employs the branch-and-bound technique combined with an iterative deepening A* (IDA*) search strategy (Korf 1985) and several tree preprocessing heuristics. The algorithm achieves good computational performance by exploiting the fact that the bid space is likely to be sparse in many situations, i.e., that the bidders will only make bids on relatively few itemsets (out of $2^N - 1$ possible itemsets). As reported in Sandholm (2002), the worst-case computational complexity for each IDA* iteration is $O(\tilde{k} \cdot \tilde{N}^2 \cdot (\tilde{k}/\tilde{N})^{\tilde{N}})$, where $\tilde{N}$ $(\tilde{N} \leq N)$ is the number of items and $\tilde{k}$ $(\tilde{k} \leq k)$ is the number of bids, respectively, that needed to be updated during this iteration. While it is difficult to obtain the precise estimations of $\tilde{N}$ and $\tilde{k}$ (these values depend on the specifics of the underlying data and can be very different for different bid sets), in the worst case the algorithm is

[6] As explained in §3.3, live bids with smaller spans incur greater computational costs during the incremental update, because such bids belong to (and can potentially affect) a larger number of subauctions. As a result, live *single-item* bids represent the worst-case scenario for the incremental update algorithm.

exponential in the number of items (N) and polynomial in the number of bids (k), as acknowledged in Sandholm (2002).

As mentioned earlier, the complexity of our algorithm is $O(2^{N-s})$ for each live bid on s number of items, or, more generally, it has a worst-case complexity of $O(k \cdot 2^N)$ in an auction with k bids. If we want to compare the two algorithms simply as winner determination algorithms (although our algorithm provides much more information along the way), by looking at the respective complexity results we can see that they differ significantly when the number of bids increases. The worst-case computational time of our approach grows *linearly* in the number of bids, while the algorithm by Sandholm (2002) is polynomial in the number of bids with a possibly significant degree ($\tilde{N}$). Obviously, the latter algorithm should be fairly efficient in some real-life scenarios where the number of bids is relatively small[7] (measured, say, in hundreds), especially because $\tilde{N} \leq N$ and $\tilde{k} \leq k$. However, in auctions where the number of bids is large, our proposed approach may be more effective.

More importantly, the goal of this research was to provide the comprehensive real-time support to auction participants. Therefore, it would be useful to compare the Sandholm (2002) algorithm and our proposed approach in terms of their capabilities of providing important auction information in real time. While none of the existing algorithms are designed to provide deadness and winning levels for each potential bid, we can at least compare our approach with the algorithm by Sandholm (2002) in terms of their ability to provide information about winning bids at any state of the auction, i.e., after each submitted bid. As mentioned before, the complexity of our incremental algorithm is $O(2^{N-s})$ for *any* live bid on s number of items, regardless of whether it is the very first bid in an auction or the thousandth, as illustrated in Table 7. We can also obtain the winning bid combination at each auction state by running the Sandholm algorithm not just at the very end of the auction, but after each submitted bid. Again, the computational complexity of such an algorithm at *each* auction state would be $O(\tilde{k} \cdot \tilde{N}^2 \cdot (\tilde{k}/\tilde{N})^{\tilde{N}})$. Because the complexity of this algorithm crucially depends on the number of bids submitted so far, after a certain number of bids[8] it will not be possible to keep recomputing the winning bids in real time. The multitude of experimental results in Sandholm (2002) confirms this, i.e., the winner determination time increases dramatically as the number of bids increases. Therefore, while we would expect the Sandholm winner determination algorithm to outperform our incremental update algorithm initially, eventually it will no longer be able to outperform our constant-per-bid-update time, as illustrated in Table 7. In this table, for the purposes of illustration we used one of the bid distributions that was also used in Sandholm (2002). More specifically, using the abovementioned CATS software, we generated bid sets of various sizes for a 25-item combinatorial auction, where all bids were three-item bids with values randomly drawn from a specified interval. Note that the incremental update for our approach was around 0.4 seconds for any bid. The Sandholm (2002) winner determination algorithm for this bid distribution takes less than 0.1 seconds to compute the winning combination (without deadness or winning levels) for 25 bids; however, the running time grows rapidly in the number of bids, rising to about 0.6 seconds for 100 bids.[9] Other researchers, such as Xia et al. (2005), have developed tighter bounds than Sandholm's (2002) heuristics using innovative linear programming formulations, and have shown that for some problems their bounds provide

Table 7 Total Incremental Update Time (in Seconds) for Live Random 3-Item Bids with Uniformly Distributed Values Generated Using CATS Software (for a 25-Item Combinatorial Auction)

Number of bids	Total update time (seconds)	Per bid update time (seconds)	Number of bids	Total update time (seconds)	Per bid update time (seconds)
100	44.864	0.449	600	240.609	0.401
200	80.944	0.405	700	275.592	0.394
300	123.291	0.411	800	312.708	0.391
400	164.129	0.410	900	351.113	0.390
500	201.628	0.403	1,000	387.437	0.387

[7] Which, as mentioned in Sandholm (2002), was an *explicit* assumption made when designing this algorithm.

[8] Naturally, the exact number depends on the specifics of the submitted bids.

[9] These results are presented in Sandholm (2002) and have been obtained on a nearly identical machine, i.e., 450 MHz Sun Ultra 80 with 1 GB of RAM.

a much faster solution as compared to Sandholm's approach. However, because the worst case for these algorithms is still bounded by the branch-and-bound technique, the comparative strengths and weaknesses of our approach remain the same. In other words, our approach is ideally suited for real-time comprehensive bidder support at each auction state, especially when the number of bids is expected to be substantial, while one-time winner determination problems for a small number of bids often can be solved efficiently by branch-and-bound techniques.

We would also like to briefly comment on the comparison between our approach and other *iterative* combinatorial auction approaches, such as iBundle (Parkes 1999) and the proxy auction approach (Ausubel et al. 2005). The essential difference between the latter approaches and our approach is that these approaches focus on developing discrete rounds of bidding (that are imposed upon bidders); at the end of each round the winner determination problem is solved and new winners are announced. Parkes (1999) provides specific prices for bundles in each round, while Ausubel et al. (2005) force bidders to bid a single bid to a proxy agent, and then an iterative auction is run using a small bid increment in each round. Our approach allows for a natural and continuous auction without any bidder restrictions and provides a much richer set of information to bidders. However, it can also be used as a winner determination mechanism with the restrictions of iBundle or proxy auctions with original convergence properties. While we do not place any assumptions on bidder behavior, such as *myopic best response* (Parkes 1999), the impact of improved information on bidder behavior is an interesting research question; we intend to pursue this using experimental economics methodologies in the future.

To summarize, the strength of our approach is that, as long as memory is available to accommodate the auction information data arrays, it can handle arbitrarily many bids while providing the bidders with comprehensive *real-time* feedback at every auction state. In addition, our approach has very *predictable* behavior because the incremental update time can be computed in advance with certainty, which is a useful feature to have in real-time environments. Therefore, our approach can be directly used to design new mercantile processes for selling combinations of items on commercial B2C and C2C sites where the number of items is typically small.

5. Conclusions and Future Work

In this paper, we develop new theoretical insights and structural properties of the bidding dynamics in combinatorial auctions. We define new metrics of interest for the bidders, such as *deadness* and *winning levels* of bids. We develop theoretical properties of these metrics and their relationships with winning bids and winner determination. Our theoretical results are based on breaking the problem down by using the concept of *subauction* to quickly identify the bids and allocation that are affected by each new bid. We also develop efficient data structures and algorithms to test the effectiveness of our theoretical results and to explore the feasibility of real-time bidder support. Our implementation essentially reduces the bidder queries to constant-time operations that can be performed with at most a couple of array lookups, making the information-gathering task very efficient and fast. The more complex task of updating the data structures (i.e., on the arrival of a new live bid) can also be conducted in milliseconds for the auction sizes that were tested. Due to the relative lack of access to real-world combinatorial auction data, we test the computational performance of our algorithm by using the CATS package provided by Leyton-Brown et al. (2000), which simulates real-world difficult data scenarios. We also test the algorithm using randomly generated data. All the experiments indicate that our implementation can provide real-time bidder support for moderate-sized auctions.

We would like to emphasize that the results of the paper follow from the fundamental theoretical analysis of bidding dynamics in combinatorial auctions, which has been largely ignored in the research literature. As this paper demonstrates, such a systematic approach can lead to significant practical improvements, such as the ability to provide real-time bidder support.

Our approach also has significant implications for business implementations in the domain of business-to-customer (B2C) and customer-to-customer (C2C) online auctions. One of the primary characteristics of most online auction mechanisms is that they are iterative multiround ascending (or descending) mechanisms. A basic requirement for generating

bidder participation in such auctions is the availability of information regarding the current state of the auction. For example, one of the key pieces of information is the minimum bid required to be winning the auction at a given point in time. While the online auctioneers have implemented many different variations of classical single-item (although some have multiple units) auctions, there are no implementations of iterative combinatorial auctions to sell multiple items to multiple bidders. We conjecture that one of the main reasons for nonavailability of iterative combinatorial auctions as a mechanism has to do with a lack of ability to provide the basic auction state information to bidders in real time. Thus, we believe our research can facilitate the introduction of a new class of auction mechanisms for B2C and C2C auctions.

Some areas of future research include exploration of special problem structures and support for XOR bids, i.e., where a bidder may be interested in one of several possible itemsets. Note, however, that XOR bids are more of a concern in sealed-bid combinatorial auctions where bidders cannot evaluate the fate of their bids themselves. In an environment with real-time information (as proposed in this paper), bidders can themselves evaluate all the itemsets that are of interest to them and place the bids on the itemset that ranks the highest based on their respective objective functions.

Finally, an NP-hard problem clearly cannot be solved efficiently without any limitations. We approach the problem from the perspective of the number of bids, i.e., because the bids can be handled very efficiently, we can support any number of bids during an auction. However, the number of items is limited to 25–30—although it is still a large improvement over the previously known complete enumeration approaches. In addition, as the technology improves in terms of memory and processing speed, we should see an improvement in the number of items as well. A key aspect of our approach is the use of subauctions to update the auctionwide information. This artifact can be exploited to create parallel algorithms, increasing the current limitation to a potentially larger number, while keeping the overall response time reasonable.

Acknowledgments
Alok Gupta's research is supported by NSF CAREER Grant IIS-0301239, but does not necessarily reflect the views of the NSF. The authors thank the workshop participants at WITS-03, Seattle, and Revolutionary Strategies and Tactics in Research Design and Data Collection for E-Business Management Research at ICEC-03, Pittsburgh, for their valuable comments and suggestions.

Appendix A. Proofs of Theoretical Results

THEOREM 1. $(\forall X \not\supseteq S(b_{k+1}))\ (WIN_{k+1}[X] = WIN_k[X])$.

PROOF. Assume $X \not\supseteq S(b_{k+1})$. Then we have $(b_{k+1} \notin B_{k+1}[X]) \Rightarrow (B_k[X] = B_{k+1}[X]) \Rightarrow (\mathbb{C}_k[X] = \mathbb{C}_{k+1}[X])$. Based on this and the definition of $WIN_{k+1}[X]$, we immediately have that $WIN_{k+1}(X) = \max_{\prec} \mathbb{C}_{k+1}[X] = \max_{\prec} \mathbb{C}_k[X] = WIN_k(X)$. □

THEOREM 2. $(\forall X \supseteq S(b_{k+1}))\ (WIN_{k+1}[X] = \max_{\prec}[WIN_k[X], \{b_{k+1}\} \cup WIN_k[X \setminus S(b_{k+1})]])$.

PROOF. Assume $X \supseteq S(b_{k+1})$. Let C_1 and C_2 be two subsets of $\mathbb{C}_{k+1}[X]$ that are defined as follows: $C_1 = \{C \in \mathbb{C}_{k+1}[X] \mid b_{k+1} \in C\}$, $C_2 = \{C \in \mathbb{C}_{k+1}[X] \mid b_{k+1} \notin C\}$. Clearly, $C_1 \cap C_2 = \varnothing$ and $C_1 \cup C_2 = \mathbb{C}_{k+1}[X]$. Therefore,

$$\begin{aligned} WIN_{k+1}[X] &= \max_{\prec} \mathbb{C}_{k+1}[X] = \max_{\prec}[C_1 \cup C_2] \\ &= \max_{\prec}\Big[\max_{\prec} C_1, \max_{\prec} C_2\Big]. \end{aligned}$$

Because $C_2 = \{C \in \mathbb{C}_{k+1}[X] \mid b_{k+1} \notin C\} = \{C \in \mathbb{C}_k[X]\} = \mathbb{C}_k[X]$, we derive $\max_{\prec} C_2$ as follows:

$$\max_{\prec} C_2 = \max_{\prec} \mathbb{C}_k[X] = WIN_k[X].$$

Furthermore, we have defined C_1 so that $(\forall C \in C_1)\ (b_{k+1} \in C)$. Based on this, $\max_{\prec} C_1$ is

$$\begin{aligned} \max_{\prec} C_1 &= \max_{\prec}\{C \in \mathbb{C}_{k+1}[X] \mid b_{k+1} \in C\} \\ &= \{b_{k+1}\} \cup \max_{\prec}\{C \setminus \{b_{k+1}\} \mid C \in \mathbb{C}_{k+1}[X], b_{k+1} \in C\} \\ &= \{b_{k+1}\} \cup \max_{\prec}\{C \mid C \in \mathbb{C}_k[X], S(C) \cap S(b_{k+1}) = \varnothing\} \\ &= \{b_{k+1}\} \cup \max_{\prec}\{C \in \mathbb{C}_k[X] \mid S(C) \subseteq X \setminus S(b_{k+1})\} \\ &= \{b_{k+1}\} \cup \max_{\prec} \mathbb{C}_k[X \setminus S(b_{k+1})] \\ &= \{b_{k+1}\} \cup WIN_k[X \setminus S(b_{k+1})]. \end{aligned}$$

Hence, we have that $WIN_{k+1}[X] = \max_{\prec}[WIN_k[X], \{b_{k+1}\} \cup WIN_k[X \setminus S(b_{k+1})]]$. □

LEMMA 1. *For any auction state k and itemsets X, Y, the following statements are true*:

1. $VL_k(X) \le VL_{k+1}(X)$;
2. $(X \subseteq Y) \Rightarrow (VL_k(X) \le VL_k(Y))$;
3. $(X \cap Y = \varnothing) \Rightarrow (VL_k(X) + VL_k(Y) \le VL_k(X \cup Y))$.

PROOF.

1. By definition, $VL_k(X) = v(WIN_k[X]) = v(\max_{\prec} \mathbb{C}_k[X]) = \max_{C \in \mathbb{C}_k[X]} v(C)$. Similarly, $VL_{k+1}(X) = \max_{C \in \mathbb{C}_{k+1}[X]} v(C)$. Because $\mathbb{C}_k[X] \subseteq \mathbb{C}_{k+1}[X]$, we have $VL_k(X) \le VL_{k+1}(X)$.

2. Similarly to (1), $VL_k(X) = \max_{C \in \mathbb{C}_k[X]} v(C)$ and $VL_k(Y) = \max_{C \in \mathbb{C}_k[Y]} v(C)$. Because $X \subseteq Y$, we have that $VL_k(X) \leq VL_k(Y)$.

3. $X \cap Y = \varnothing \Rightarrow S(WIN_k[X]) \cap S(WIN_k[Y]) = \varnothing \Rightarrow (WIN_k[X] \cup WIN_k[Y]) \in \mathbb{C}_k[X \cup Y] \Rightarrow v(WIN_k[X] \cup WIN_k[Y]) \leq v(\max_{\prec} \mathbb{C}_k[X \cup Y]) \Rightarrow VL_k(X) + VL_k(Y) \leq VL_k(X \cup Y)$. □

THEOREM 3. *Given auction state k and bid $b \in B_k$ such that $S(b) = X$, we have: $b \in LIVE_k \Leftrightarrow b \in WIN_k[X]$.*

PROOF. $\boxed{\Rightarrow}$ Assuming $b \in LIVE_k$, by definition we have that $\exists B_l \supseteq B_k$ such that $b \in WIN_l$. Suppose otherwise, $b \notin WIN_k[X]$, and therefore, $\{b\} \prec WIN_k[X]$. Let W be the following bid combination: $W = (WIN_l \backslash \{b\}) \cup WIN_k[X]$. Since $(WIN_l \backslash \{b\}) \cap WIN_k[X] = \varnothing$, W is a valid bid combination, i.e., $W \in \mathbb{C}_l$. Furthermore, the $\{b\} \prec WIN_k[X]$ order on bid combinations implies that $\{b\} \cup (WIN_l \backslash \{b\}) \prec WIN_k[X] \cup (WIN_l \backslash \{b\})$. Hence, we have $WIN_l \prec W$, where $W \in \mathbb{C}_l$. Therefore, WIN_l cannot be a winning combination at stage l—a contradiction. $\boxed{\Leftarrow}$ Assuming $b \in WIN_k[X]$, we have that $\{b\} = WIN_k[X]$ and, therefore, $v(b) = VL_k(X)$. Consider auction state $(k+1)$ and choose a new bid b_{k+1} such that $S(b_{k+1}) = \mathcal{I} \backslash X$ and $v(b_{k+1}) > VL_k(\mathcal{I}) - VL_k(X)$. We can rewrite the previous inequality as: $v(WIN_k[\mathcal{I}]) < v(b_{k+1}) + v(WIN_k[\mathcal{I} \backslash S(b_{k+1})])$. By directly applying Theorem 2, we then have that $WIN_{k+1} = \{b_{k+1}\} \cup WIN_k[X]$. Because $b \in WIN_k[X]$, we have that $b \in WIN_{k+1}$. Then, by definition, $b \in LIVE_k$. □

COROLLARY 3A. $LIVE_k = \bigcup_{X \subseteq \mathcal{I}} WIN_k[X]$.

PROOF. Suppose $b \in LIVE_k$. From Theorem 3 we have that $b \in WIN_k[S(b)]$, and, since $S(b) \subseteq \mathcal{I}$, we have $b \in \bigcup_{X \subseteq \mathcal{I}} WIN_k[X]$. Conversely, if $b \in \bigcup_{X \subseteq \mathcal{I}} WIN_k[X]$, then there must exist subauction X, $X \supseteq S(b)$, such that $b \in WIN_k[X]$. From Lemma 3, we have that $b \in WIN_k[S(b)]$. Furthermore, by applying Theorem 3 we get $b \in LIVE_k$. □

COROLLARY 3B. *Given auction state k and new bid b_{k+1}, such that $S(b_{k+1}) = X$, the following is true: $b_{k+1} \in DEAD_{k+1} \Leftrightarrow v(b_{k+1}) \leq VL_k(X)$.*

PROOF. $\boxed{\Rightarrow}$ If $b_{k+1} \in DEAD_{k+1}$, then from Theorem 3 we have that $b_{k+1} \notin WIN_{k+1}[X]$ and, therefore (based on Theorem 2), $WIN_{k+1}[X] = WIN_k[X]$. Consequently, $b_{k+1} \prec WIN_{k+1}[X]$, which implies $v(b_{k+1}) \leq VL_{k+1}(X) = VL_k(X)$. $\boxed{\Leftarrow}$ Conversely, if $v(b_{k+1}) \leq VL_k(X)$, we have that $b_{k+1} \prec WIN_k[X]$ because $WIN_k[X]$ precedes b_{k+1} chronologically. Therefore, based on Theorem 2, $b_{k+1} \notin WIN_{k+1}[X]$ and, from Theorem 3, $b_{k+1} \in DEAD_{k+1}$. □

THEOREM 4. *Given auction state k and new bid b_{k+1}, such that $S(b_{k+1}) = X$: $b_{k+1} \in WIN_{k+1} \Leftrightarrow v(b_{k+1}) > VL_k(\mathcal{I}) - VL_k(\mathcal{I} \backslash X)$.*

PROOF. From Theorem 2 we have: $b_{k+1} \in WIN_{k+1}$ if and only if $WIN_k \prec \{b_{k+1}\} \cup WIN_k[\mathcal{I} \backslash X]$. Based on the definition of strict total order $\prec$ and taking into account that combination WIN_k chronologically precedes bid combination $\{b_{k+1}\} \cup WIN_k[\mathcal{I} \backslash X]$, we have that $VL_k(\mathcal{I}) < v(b_{k+1}) + VL_k(\mathcal{I} \backslash X)$. □

THEOREM 5. *For any auction state k and itemsets X, Y, the following statements are true*:

1. $DL_k(X) \leq WL_k(X)$;
2. $DL_k(\mathcal{I}) = WL_k(\mathcal{I})$;
3. $WL_k(X) = DL_k(\mathcal{I}) - DL_k(\mathcal{I} \backslash X)$;
4. $DL_k(X) = WL_k(\mathcal{I}) - WL_k(\mathcal{I} \backslash X)$;
5. $DL_k(X) \leq DL_{k+1}(X)$;
6. $X \subseteq Y \Rightarrow DL_k(X) \leq DL_k(Y)$;
7. $X \subseteq Y \Rightarrow WL_k(X) \leq WL_k(Y)$;
8. $X \cap Y = \varnothing \Rightarrow DL_k(X \cup Y) \geq DL_k(X) + DL_k(Y)$;
9. $b \in LIVE_k$, s.t. $S(b) = X \Rightarrow DL_k(X) = v(b)$;
10. $b \in WIN_k$, s.t. $S(b) = X \Rightarrow WL_k(X) = v(b)$.

PROOF.

1. Based on Lemma 1, $VL_k(\mathcal{I}) \geq VL_k(\mathcal{I} \backslash X) + VL_k(X)$. Hence, $VL_k(X) \leq VL_k(\mathcal{I}) - VL_k(\mathcal{I} \backslash X)$ and, from definitions of $DL_k(X)$ and $WL_k(X)$, we have $DL_k(X) \leq WL_k(X)$.

2. By definition, $DL_k(\mathcal{I}) = VL_k(\mathcal{I})$ and $WL_k(\mathcal{I}) = VL_k(\mathcal{I}) - VL_k(\varnothing) = VL_k(\mathcal{I})$.

3. Immediately from definitions, i.e., $WL_k(X) = VL_k(\mathcal{I}) - VL_k(\mathcal{I} \backslash X) = DL_k(\mathcal{I}) - DL_k(\mathcal{I} \backslash X)$.

4. Immediately from (3), only replace X by $\mathcal{I} \backslash X$ and $DL_k(\mathcal{I})$ by $WL_k(\mathcal{I})$ (based on 2).

5. From Lemma 1, $VL_k(X) \leq VL_{k+1}(X)$. Hence, $DL_k(X) \leq DL_{k+1}(X)$.

6. From Lemma 1, $X \subseteq Y \Rightarrow VL_k(X) \leq VL_k(Y)$. Hence, $DL_k(X) \leq DL_k(Y)$.

7. Similarly to (6), $X \subseteq Y \Rightarrow \mathcal{I} \backslash Y \subseteq \mathcal{I} \backslash X \Rightarrow VL_k(\mathcal{I} \backslash Y) \leq VL_k(\mathcal{I} \backslash X) \Rightarrow VL_k(\mathcal{I}) - VL_k(\mathcal{I} \backslash X) \leq VL_k(\mathcal{I}) - VL_k(\mathcal{I} \backslash Y) \Rightarrow WL_k(X) \leq WL_k(Y)$.

8. From Lemma 1, $X \cap Y = \varnothing \Rightarrow VL_k(X \cup Y) \geq VL_k(X) + VL_k(Y) \Rightarrow DL_k(X \cup Y) \geq DL_k(X) + DL_k(Y)$.

9. From Theorem 3 we have that $b \in WIN_k[X]$ or, in other words, $WIN_k[X] = \{b\}$. Hence, $v(b) = VL_k(X)$ and, therefore, $v(b) = DL_k(X)$.

10. $b \in WIN_k \Rightarrow WIN_k = \{b\} \cup WIN_k[\mathcal{I} \backslash X] \Rightarrow VL_k(\mathcal{I}) = v(b) + VL_k(\mathcal{I} \backslash X)$. Hence, $v(b) = VL_k(\mathcal{I}) - VL_k(\mathcal{I} \backslash X)$ and, therefore, $v(b) = WL_k(X)$. □

LEMMA 2. *In a combinatorial auction of size N:*

1. *The number of live bids at any auction state cannot be greater than $2^N - 1$;*
2. *$2^N - 1$ is a tight upper bound, i.e., it is possible to have an auction state where there are exactly $2^N - 1$ live bids.*

PROOF.

(1) A set of size N can have $2^N - 1$ different nonempty subsets. Therefore, in a combinatorial auction of size N there can be $2^N - 1$ different bid spans (corresponding to each nonempty subset). Obviously, there can be only one live bid with given span X. (If there are multiple bids submitted with the same span, only the one with the largest value can possibly be a live bid. If there are several bids with the same span and the same largest value, only the ear-

liest of them can possibly be a live bid.) Therefore, $|LIVE_k| \leq 2^N - 1$ for any auction state k.

(2) Suppose that exactly one bid has been submitted on each possible itemset (hence, there are $2^N - 1$ bids), where the value of the bid on itemset X was set as $2 \cdot |X| - 1$. Then the value of any bid combination C can be expressed as: $v(C) = \sum_{b \in C} v(b) = \sum_{b \in C} 2 \cdot |S(b)| - 1 = 2 \cdot |S(C)| - |C|$. By maximizing $2 \cdot |S(C)| - |C|$, we derive that the current winning combination consists of a single bid on the whole auction set and has value $2 \cdot N - 1$. It is easy to see that all these $2^N - 1$ bids are live. Suppose, otherwise, that there exists bid b (where $S(b) = X$ and $v(b) = 2 \cdot |X| - 1$) that is dead. Choose a brand new bid b', such that $S(b') = \mathcal{I} \backslash X$ and $v(b') > 2 \cdot N - 2 \cdot |X|$. Then, bid combination $C' = \{b, b'\}$ is a new winning combination, because its value $v(C) = v(b) + v(b') > 2 \cdot |X| - 1 + 2 \cdot N - 2 \cdot |X| = 2 \cdot N - 1$. Because $b \in C'$, by definition b is live—a contradiction. □

Lemma 3. *For any bid $b \in B_k$ and any subauction $X \supseteq S(b)$: $b \in WIN_k[X] \Rightarrow b \in WIN_k[S(b)]$.*[10]

Proof. Assume that $X \neq S(b)$ (the case $X = S(b)$ is straightforward). Suppose, otherwise, that $b \notin WIN_k[S(b)]$ and, therefore, $\{b\} \prec WIN_k[S(b)]$. Consider $W = (WIN_k[X] \backslash \{b\}) \cup WIN_k[S(b)]$. Because $S(WIN_k[S(b)]) \subseteq S(b)$, W is a valid bid combination for subauction X, i.e., $W \in \mathbb{C}_k[X]$. Furthermore, $\{b\} \prec WIN_k[S(b)] \Rightarrow \{b\} \cup (WIN_k[X] \backslash \{b\}) \prec WIN_k[S(b)] \cup (WIN_k[X] \backslash \{b\}) \Rightarrow WIN_k[X] \prec W$, where $W \in \mathbb{C}_k[X]$. Hence, $WIN_k[X]$ is not a winning bid combination—a contradiction. □

References

Ausubel, L., P. Milgrom. 2002. Ascending auctions with package bidding. *Frontiers Theoret. Econom.* **1**(1). Available from The Berkeley Electronic Press. http://www.bepress.com/bejte/frontiers/vol1/iss1/art1/.

Ausubel, L., P. Cramton, P. Milgrom. 2006. The clock-proxy auction: A practical combinatorial auction design. P. Cramton, Y. Shoham, R. Steinberg, eds. *Combinatorial Auctions*, ch. 5. Forthcoming, MIT Press, Boston, MA. www.cramton.umd.edu/papers2000-2004/ausubel-cramton-milgrom-the-clock-proxy-auction.pdf.

Banks, J. S., J. O. Ledyard, D. Porter. 1989. Allocating uncertain and unresponsive resources: An experimental approach. *RAND J. Econom.* **20**(1) 1–25.

Bitner, J., G. Ehrlich, E. Reingold. 1976. Efficient generation of the binary reflected gray code and its applications. *Comm. ACM* **19**(9) 517–521.

Brewer, P. J., C. R. Plott. 1996. A binary conflict ascending price (BICAP) mechanism for the decentralized allocation of the right to use railroad tracks. *Internat. J. Indust. Organ.* **14**(6) 857–886.

de Vries, S., R. Vohra. 2003. Combinatorial auctions: A survey. *INFORMS J. Comput.* **15**(3) 284–309.

Hevner, A., S. March, J. Park, S. Ram. 2004. Design science in information systems research. *MIS Quart.* **21**(1) 75–105.

Hudson, B., T. Sandholm. 2002. Effectiveness of preference elicitation in combinatorial auctions. AAMAS-02 Workshop on Agent-Mediated Electronic Commerce. http://www-2.cs.cmu.edu/~sandholm/elicitation.CMU-CS-02-124.pdf.

Kelly, F., R. Steinberg. 2000. A combinatorial auction with multiple winners for universal service. *Management Sci.* **46**(4) 586–596.

Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence* **27**(1) 97–109.

Leyton-Brown, K., M. Pearson, Y. Shoham. 2000. Towards a universal test suite for combinatorial auction algorithms. *Proc. ACM Conf. Electronic Commerce*, ACM Press, New York.

Nisan, N. 2000. Bidding and allocation in combinatorial auctions. *Proc. ACM Conf. Electronic Commerce*, ACM Press, New York. http://www. cs.huji.ac.il/~noam/auctions.pdf.

Parkes, D. C. 1999. iBundle: An efficient ascending price bundle auction. *Proc. ACM Conf. Electronic Commerce*, ACM Press, New York, 148–157.

Pekec, A., M. H. Rothkopf. 2003. Combinatorial auction design. *Management Sci.* **49**(11) 1485–1503.

Rothkopf, M. H., A. Pekec, R. M. Harstad. 1998. Computationally manageable combinatorial auctions. *Management Sci.* **44**(8) 1131–1147.

Sandholm, T. 1999. An algorithm for optimal winner determination in combinatorial auctions. *Proc. IJCAI-99*. Morgan Kaufman, Stockholm, Sweden, 542–547.

Sandholm, T. 2002. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence* **135** 1–54.

Tennenholtz, M. 2000. Some tractable combinatorial auctions. *Proc. 17th National Conf. Artificial Intelligence*. AAAI Press, Austin, TX, 98–103.

Xia, M., J. Stallaert, A. Whinston. 2005. Solving the combinatorial double auction problem. *Eur. J. Oper. Res.* **164** 239–251.

[10] This lemma is only used in the proof of Corollary 3a.