# A Comparison of Two Algorithms for Multi-unit $k$-Double Auctions

Shengli Bao    Peter R. Wurman

North Carolina State University
Department of Computer Science
Raleigh, NC 27695-7535 USA
baoshengli2000@yahoo.com    wurman@ncsu.edu

April 1, 2003

**Abstract**

We develop two algorithms to manage bid data in flexible, multi-unit double auctions. The first algorithm is a multi-unit extension of the 4-HEAP algorithm, and the second is a novel algorithm based on the Internal Path Reduction tree. To facilitate the generation of price quotes, we enhance the IPR tree algorithm to maintain information about the current $M$th and $(M+1)$st units. Our experiments show that when bids are relatively unordered, AUC-IPR outperforms 4-HEAP by a factor of about 2.5. However, under some scenarios in which bids are received in order, AUC-IPR is not always better than 4-HEAP.

# 1 Introduction

Auctions that simultaneously mediate among multiple buyers and sellers—generically referred to as *double auctions*—are most widely used in financial markets [2]. Particular double auctions may differ in how prices are set, when price quotes are generated, and when trades are computed [10]. Many stock markets open with prices that are established by operating a *clearinghouse* (a double auction with a fixed closing time) immediately before

public trading begins [4]. During the trading day, stock markets operate as *continuous* double auctions in which each new bid may engender a transaction. As e-commerce continues to permeate business-to-business interactions, and more unusual commodities are traded [5], we expect to see further diversity in the types of double auctions being employed.

In many applications, auctions will be required to handle a large number of bids. In such cases, the computational performance of the algorithms that manage bid data will be an important aspect of the design of the auction system. In this paper we examine two algorithms for bid management. The first is an extension of the 4-HEAP algorithm [10], and the second is a new algorithm based on the IPR tree [3], which we refer to as AUC-IPR.[1] Both algorithms are designed for use in flexible auction systems, that is, auction software designed to support many variations of the double auction format. Thus, our investigation focuses on the general case, and we recognize that special purpose algorithms may perform better for restricted auctions.

The general double auction admits possibly multi-unit bids to buy or sell a commodity. We assume that all bids can be partially satisfied, that is, an offer to buy (or sell) $x$ units is an expression that the bidder is willing to trade between zero and $x$ units, inclusively.[2] Further, we assume the auction computes clearing prices and price quotes using the $k$-double auction policy [6], although the algorithms developed could be modified to handle some other types of double auctions. The $k$-double auction policy finds the range of equilibrium prices for the auction, and then computes the clearing price as the lower bound plus a proportion (given by $k$) of the range. Section 2 reviews the $k$-double auction policy.

An auction has three core activities [11]: to admit bids, to generate price information, and to clear. These core activities define the tasks that a general auction algorithm must handle.

- **Insert/Remove Bid**—When a new bid is received, and the auction system verifies that it satisfies whetever bidding rules exist, it must be inserted into the auction's bid data structures. Similarly, when a bid is withdrawn, it must be removed from the data structures.
- **Compute Quote**—The auction will generate price quote information according to some schedule. Much of the complexity of the algorithms we present in this paper is due to the need to maintain quote information across insert and remove operations.

[1]This work is based on the Masters thesis of the first author [1].

[2]Computing the set of winning bids when indivisible bids are admitted is a packing problem, and a different class of algorithms is needed.

- **Clear**—At designated times, the auction will compute exchanges between the buyers and sellers, notify the participants, and remove the winning bids from the data structures.

Both the 4-HEAP and AUC-IPR algorithms are based on well-known data structures. Through the use of fairly straightforward enhancements, we are able to maintain the bookkeeping information necessary to perform the three tasks. In Section 2 we review the $k$-double auction. Sections 3 and 4 present the 4-HEAP and AUC-IPR algorithms, respectively. In Section 5 we report on our empirical investigation of the relative performance of the two algorithms under different assumptions about the schedules of the quote and clear events. Finally, Section 6 summarizes the results and the implications for auction software designers.

# 2 $k$-Double Auctions

The $k$-double auction policy is a generalization of the classic first-price and second-price auctions. When a single item is for sale, the first-price auction assigns it to the highest bidder at a price equal to the winner's bid. The second-price auction [8] assigns the item to the highest bidder but charges a price equal to the second highest bidder's offer.

When the scenario involves multiple buyers and sellers, each offering to buy or sell a single unit, we compute the $M$th and $(M + 1)$st prices, where $M$ is the number of sell bids. Throughout the paper we assume that a total order can be imposed on all the bids. This is commonly accomplished using price as the principal priority measure, and using bid quantity or bid placement time to break ties. Conceptually, finding the $M$th and $(M + 1)$st bids is simply a matter of sorting the bids in descending order, and identifying the $M$th and $(M + 1)$st items in the list. The prices between the $M$th and $(M + 1)$st bids (inclusively) represent the range of prices for which supply balances demand. At prices in the range, the number of buyers willing to buy at that price equals the number of sellers willing to sell, with the caveat that when $M$th= $(M + 1)$st, one side or the other may have some participants who lose on tie-breaking criteria. It is important to note that this process of identifying the equilibrium price range works regardless of the relative position of the buyers and sellers in the list.

The $k$-double auction computes a clearing price that is a ratio of the two boundary prices. Specifically, if $p_M$ is the $M$th-price, and $p_{M+1}$ is the $(M + 1)$st-price, the $k$-double auction will set $p = kp_{M+1} + (1 - k)p_M$, $0 \leq k \leq 1$. Furthermore, the $M$th and $(M + 1)$st prices delineate the set

| Sell Bids | Buy Bids |
|---|---|
| Sell 1 @ $23 | Buy 3 @ $24 |
| Sell 1 @ $21 | Buy 1 @ $22 |
| Sell 3 @ $20 | Buy 3 @ $19 |
| Sell 3 @ $17 | Buy 4 @ $15 |
| Sell 2 @ $16 | Buy 1 @ $13 |
| Sell 1 @ $14 | Buy 1 @ $10 |
| Sell 1 @ $12 | |
| Sell 3 @ $11 | |

Table 1: An example set of bids. Sell bids below the line, and buy bids above the line are currently winning.

of currently winning bids, which we refer to as the *transaction set*. Again, modulo ties at the boundaries, buyers at or above the $M$th-price would purchase an item if the auction cleared, and sellers at or below the $(M+1)$st-price would sell an item. It follows that the $M$th-price and $(M+1)$st-price constitute exactly the information that is typically provided to participants in the form of price quotes. The $M$th-price is the *ask quote* and informs a potential buyer of the minimum that she would have to offer to be certain to enter the current transaction set. Symmetrically, the *bid quote*, equal to the $(M+1)$st-price, informs a potential seller the maximum that he would be able to offer to become a current winner.

The procedure can be extended to multi-unit, partially satisfiable bids simply by treating each unit offered as a separate bid. In this case, $M$ is the number of units offered for sale, and $N$ is the total number of units in all bids. The $M$th-price ($(M+1)$st-price) is set by the price on the $M$th ($(M+1)$st) highest unit. Table 1 presents an example set of bids that we will use throughout the paper. Following the procedure for finding the $M$th and $(M+1)$st prices, we first determine that $M = 15$ units are for sale. The position of the fifteenth highest bid is the third unit of the offer to sell three units at $17. The sixteenth bid is the first unit of the offer to sell two units at $16. The line in each column separates bids that are at or above the $M$th-price from those that are at or below the $(M+1)$st-price.

# 3 4-Heap

The 4-HEAP algorithm takes its name from the fact that the bids are organized into four heap data structures, representing the currently winning buy

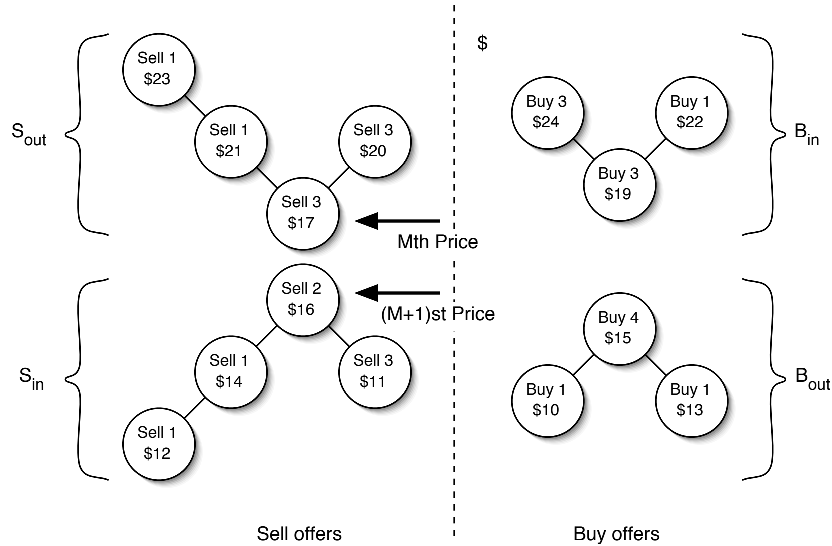


Figure 1: A diagram of the four heaps for the example problem.

offers, the currently winning sell offers, the currently non-winning buy offers, and the currently non-winning sell offers. We denote the four heaps as $\mathrm{B}_{in}$, $\mathrm{S}_{in}$, $\mathrm{B}_{out}$, and $\mathrm{S}_{out}$, respectively. $\mathrm{B}_{in}$ and $\mathrm{S}_{out}$ are $\min$ heaps, and $\mathrm{B}_{out}$ and $\mathrm{S}_{in}$ are $\max$ heaps. Naturally, the minimal bid in $\mathrm{B}_{in}$ must be at least as great as the maximal bid in $\mathrm{B}_{out}$, and the minimal bid in $\mathrm{S}_{out}$ must be at least as great as the maximal bid in $\mathrm{S}_{in}$. The heaps have the further properties that the number of units in $\mathrm{B}_{in}$ and $\mathrm{S}_{in}$ must be the same, and the minimal member of $\mathrm{S}_{out}$ is strictly less than the maximal member of $\mathrm{B}_{out}$. Figure 1 illustrates the heap configuration for the bids in Table 1.

An important benefit of the 4-HEAP algorithm is that the $M$th and $(M+1)$st prices are easily calculated from the heaps. Specifically, the $M$th-price is the $\min$ of $\mathrm{B}_{in}$ and $\mathrm{S}_{out}$, and the $(M+1)$st-price is the maximal members of $\mathrm{B}_{out}$ and $\mathrm{S}_{in}$, both of which can be easily computed from the values of the highest priority nodes in each heap. In addition, clearing the auction is simply a matter of deconstructing the $\mathrm{B}_{in}$ and $\mathrm{S}_{in}$ heaps, and leaving the other two heaps intact.

When a bid is received, the heaps must be adjusted to maintain the 4-HEAP properties. The algorithm originally proposed by Wurman, Walsh, and Wellman [10], was worked out for single-unit bids, and suggested an inefficient method for handling multi-unit bids. Later, Walsh [9] suggested a technique to enable the algorithm to handle multi-unit bids more efficiently,

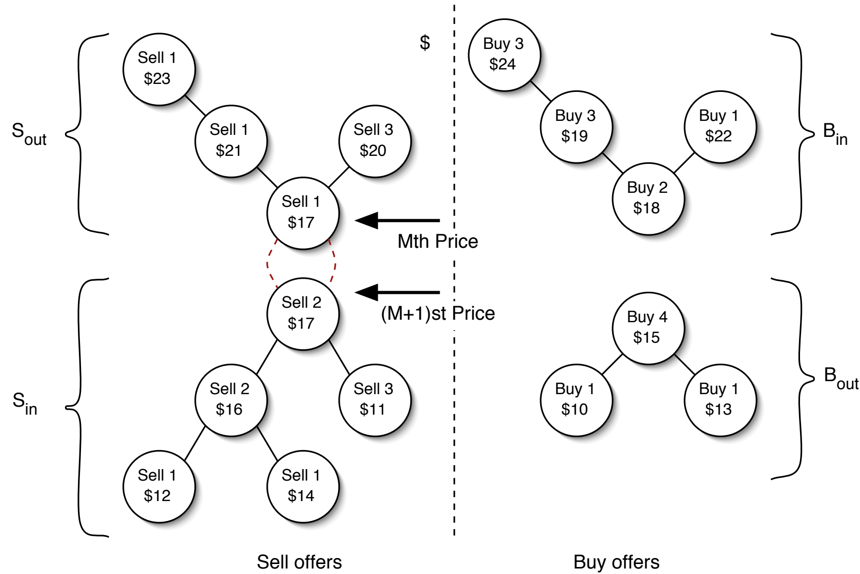


Figure 2: The four heaps adjusted after inserting {Buy 2 @ \$18}.

which we describe here. Due to space limitations, we cannot provide the pseudo code, but limit ourselves to a brief exposition. We describe the insertion process; bid removal is symmetric and has the same computational complexity.

When a new bid is inserted, the algorithm first determines whether it (a) should be inserted directly into an OUT heap, (b) should be matched with bid(s) in the complementary OUT heap and all implicated bids moved to IN heaps, or (c) displace some bids from the appropriate IN heap. In both cases (b) and (c), the new bid may need to be split across the IN and OUT heaps during this process. Split bids can be reassembled if the component parts are returned to the same heap, thus, the algorithm will never have more than one split bid. Note that steps (b) and (c) may result in several nodes being moved between heaps; in the worst case, $q$ nodes will have to be moved, where $q$ is the quantity associated with the new bid.

Consider the effects of inserting a buy bid for two units at \$18 in the problem presented in Table 1. To properly adjust the heaps, we must create a new node and insert it into $\mathrm{B}_{in}$. We must split the {Sell 3 @ \$17} into a two unit offer inserted into $\mathrm{S}_{in}$, and a one unit offer remaining in $\mathrm{S}_{out}$. The final configuration is shown in Figure 2. The dotted line between the two nodes representing the sell offer for \$17 identify the bid as split.

The computational complexity of inserting a new bid is $O((2q+1)\ln N)$.[3] This result follows from the fact that each node moved from heap $H_1$ to heap $H_2$ requires one $O(\lg N)$ operation to restore $H_1$, and one $O(\lg N)$ to push the node onto $H_2$. A price quote can be generated in constant time by simply computing the $M$th- and $(M+1)$st-prices as mentioned above. Setting aside the issue of *how* bids are matched, and focusing only on separating the winning bids from the non-winning bids, clearing takes $O(N)$ time in 4-HEAP because the two IN heaps can be directly disassembled. Policies that construct matches based on features (quantity, time, or price) of the winning bids are beyond the scope of this study.

While the big-O performance of the 4-HEAP algorithm is, in a sense, as good as we can expect, we investigated algorithms that may be able to improve the actual performance by reducing the number of $\ln n$ operations. In particular, the 4-HEAP algorithm has the drawback that its typical performance is close to its worst case performance because of the manner in which objects are popped and pushed onto the heap. In 4-HEAP, when a bid is moved from an IN heap to an OUT heap, or vice-versa, it is certain to require two operations (a pop then a push) that require exactly $\ln N$ time. The next section presents an algorithm that attempts to avoid this seemingly unnecessary computation.

# 4 auction-IPR

There are many varieties of balanced and semi-balanced binary trees, many of which would have been suitable for our task. We have selected to base our algorithm on the Internal Path Reduction (IPR) tree [3] because its height characteristics are the same as the AVL tree, but it produces slightly more compact structures. The *internal path* is the sum of the path lengths of all the nodes measured from the root node. On average, the IPR tree algorithm produces an internal path about 10% smaller than the AVL tree algorithm. In addition, the IPR tree has a tunable parameter, $c$, that determines the acceptable level of imbalance.

IPR trees are re-balanced when one branch becomes sufficiently more populated than another, where $c$ is the threshold value. To re-balance the tree, a rotation operation is applied to promote the more populous branch. Figure 3 illustrates a possible configuration of the IPR tree for the bids in Table 1 and Figure 4 shows the tree after the insertion of the bid {Buy 2 @ \$18} (with $c = 1$). A double rotation on the top node was required because the

[3] Removing a bid is symmetric with insertion, so hereafter we ignore it.

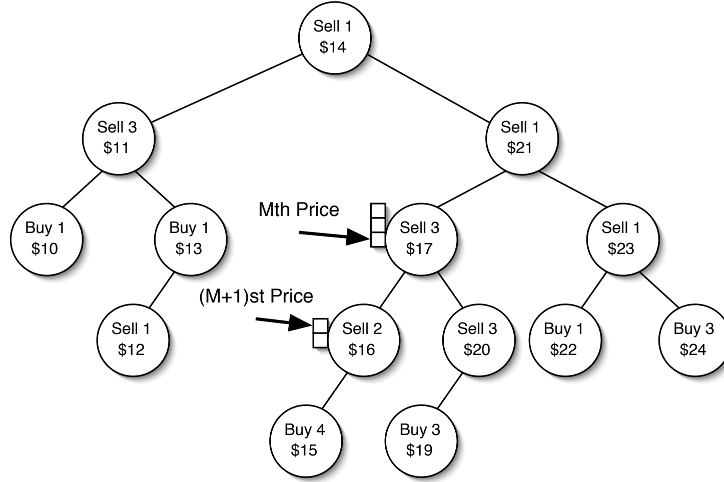


Figure 3: An IPR tree configuration for the bids in Table 1.

right side has become too populated. Details of the algorithm, and the bookkeeping information needed to implement it, are available in textbooks (e.g., [7]).

Clearly, the IPR tree can be used to maintain the bids in a fully sorted order with a basic $O(\lg N)$ insertion time. However, in order to generate a price quote, it would be necessary to traverse the entire tree to determine the $M$th and $(M+1)$st bids. The AUC-IPR algorithm enhances the basic IPR tree algorithm with the information necessary to track the position of the $M$th and $(M+1)$st bids in an efficient manner.

The first step is to maintain two pointers, one to keep track of the $M$th unit, and a second to keep track of the $(M+1)$st unit. The small squares adjoining selected nodes in Figures 3 and 4 represent the individual units, assigned an arbitrary ordering with "greatest" value on top of stack.

Now, it is simply a matter of moving the pointers as bids are inserted and removed. In particular, we may need to move the pointers if we receive a sell offer at or below the $(M+1)$st bid, or a buy offer at or above the $M$th bid. In the example, after the insertion and rebalancing illustrated in Figure 4, both pointers must be moved up two units. The $M$th pointer moves from the bottom to the top unit of $\{$Sell 3 @ \$17$\}$, and the $(M+1)$st pointer moves from its position at the top of $\{$Sell 2 @ \$16$\}$ to the second unit in $\{$Sell 3 @ \$17$\}$.[4]

[4]In practice, we do not enumerate the units in bids. Instead, we simply keep track of which

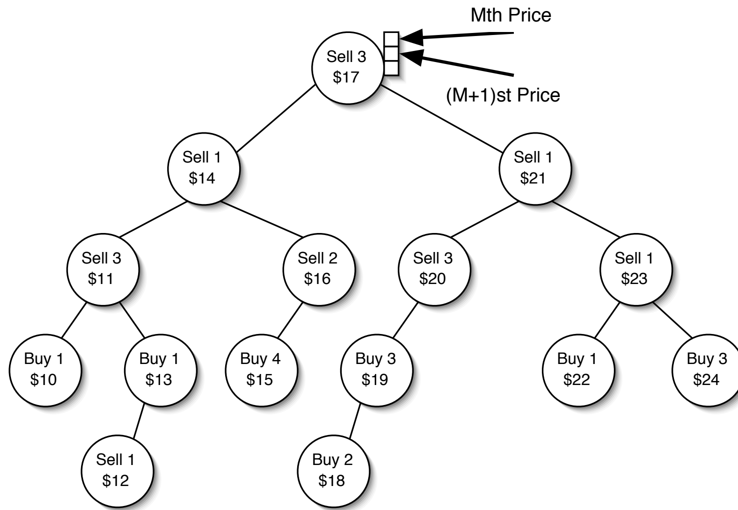


Figure 4: The IPR tree adjusted after inserting {Buy 2 @ $18}.

We could simply walk the tree the appropriate number of units, but in the worst case this could involve traversing the entire internal path. To see why, consider a tree with one thousand single-unit sell offers into which we insert a single buy offer for one thousand and one units with a price greater than all of the sell offers. Initially, the $M$th-price is set by the lowest sell offer, and the $(M+1)$st-price is undefined. After the buy offer is inserted, the $M$th- and $(M+1)$st-price is set by the buy offer. In the IPR tree, the $M$th pointer begins at one end of the tree and must propagate to the other end. Walking up and down the tree one unit at a time is clearly inefficient.

The AUC-IPR algorithm tracks the number of units on each child branch of each node. When it is necessary to move the $M$th and $(M+1)$st pointers by $q$ units, the algorithm uses these recorded values to determine when it is possible to avoid descending a branch. If a subtree attached as a child to node $n$ contains $q'$ units, where $q' < q$, and there is more of the tree to explore, the algorithm moves on to the node it would search after the subtree and considers $q'$ units already accounted for. If $q' > q$, the subtree must be explored because it contains the position of the new $M$th or $(M+1)$st pointer. In the worst case, the algorithm will have to traverse $2 \ln N$ nodes, but we expect it to be significantly less in practice.

In order to properly maintain the unit counters on the subtrees, we must enhance the insert and rotation operators. It is a simple matter to increment

unit is the focus as part of the pointer record.

the appropriate counters was we search the tree for the correct location to insert a new node. It is also relatively straightforward to update the number of units on each subtree under the two rotation schemes.

Unlike the 4-HEAP algorithm, clearing the IPR tree is costly. We implemented a fairly straightforward method for clearing the auction. AUC-IPR traverses the tree and inspects the bid at each node. If the bid should clear, the algorithm removes the node, an action that may require that the tree be re-balanced. If the node is being removed and it contains either the $M$th or $(M+1)$st unit, the pointer must be adjusted. The $(M+1)$st pointer can simply be moved to the previous node (which has already been visited and retained). The $M$th pointer must be "carried" by the algorithm as it continues to process nodes until a node is found which is being retained (i.e., a sell offer above the $M$th price). It is also necessary to update the counters in a node's ancestors as it is removed.

The insert and withdraw actions require $O(\lg N)$ time to insert a node, $O(\lg N)$ time to re-balance the tree, and $O(2 \ln n)$ time to move each pointer, for a total worst case complexity of $O(6 \ln N)$. The price quote action is constant time $O(1)$. The clear action takes $O(2N \lg N)$ time.

# 5 Performance Comparison

The purpose of this investigation is to compare the performance of 4-HEAP with AUC-IPR. We implemented the two algorithms in C++ on desktop UNIX workstations, and measured the performance of the algorithms in seconds. The following experiments were designed to study the impact of the number, distribution, and size of bids on algorithm performance. Experiments ranged in size from 10,000 to 90,000 bids; although this is a fairly big range, a balanced tree can hold all of the bids in 8–9 ply. Unless otherwise stated, each bid's quantity was drawn uniformly from the integers [1,10].

Figure 5 details the running time of the two algorithms when inserting from 10,000 to 90,000 bids. Experimentally, the IPR tree algorithm is about 2.6 times faster than the 4-Heap algorithm. This matched our intuition that it would be possible to reduce the number of operations by about about two-thirds.

Table 2 shows the effect of increasing the range from which bid quantities are drawn. The first four cases are uniform distributions with the indicated ranges, while the fifth case is based on an exponential distribution of bid sizes. In each test case, 10,000 bids are generated.

Interestingly, the run time of the 4-HEAP algorithm varies only 10%

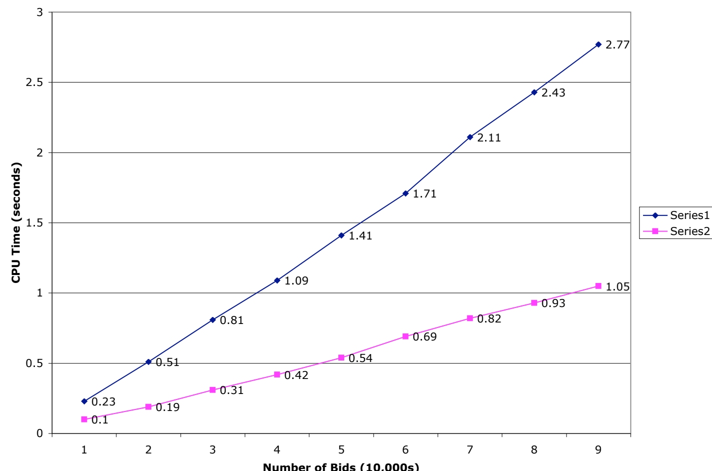


Figure 5: The run times of the two algorithms for pure insertion.

across the five test cases. The AUC-IPR algorithm is more sensitive to the bid size, but performs better across all of the test cases. We expected the run time of both algorithms to increase as the quantity range increases because the algorithm is more likely to encounter situations in which the position of the $M$th and ($M+1$)st must be moved many positions in the queue. Surprisingly, the run time of 4-HEAP, which in the worst case analysis has an explicit dependence on quantity, seems on average to perform rather steadily. In hindsight, this steady performance is to be expected. Given a random value drawn from a uniform distribution, one would expect to have to make 1.7 draws from the same distribution to draw a total value that is greater than or equal to the first draw. Thus, in the 4-HEAP algorithm, we expect to insert the new node (one $\ln N$ operation), and half the time need to readjust the other heaps with 1.7 pop-push pairs. Thus, we should expect to require $(1 + 1.7) \ln N$ operations. The AUC-IPR algorithm, on the other hand, expects to make one insert and then move the two pointers 1.7 (essentially negligible) positions in the tree. Thus, we should expect 4-HEAP to take 2.7 $lnN$ operations, compared to AUC-IPR's 1 $lnN$ operation.

In Table 2, insertion in the AUC-IPR algorithm shows dependence on

| **Bid Quantity** | **4-Heap** | **Auc-IPR** |
|---:|---|---|
| 1 | 0.229 | 0.085 |
| 1-100 | 0.233 | 0.101 |
| 1-500 | 0.246 | 0.177 |
| 1-1000 | 0.253 | 0.205 |
| Exponential | 0.227 | 0.125 |

Table 2: Time (in seconds) to insert 10,000 bids with valuations drawn from the indicated distributions.

| **Clear Interval** | **4-Heap** | **Auc-IPR** |
|---:|---|---|
| 1000 | 0.235 | 0.082 |
| 5000 | 0.236 | 0.076 |
| 10000 | 0.225 | 0.071 |

Table 3: Time (in seconds) to insert 90,000 bids with periodic clears.

bid quantity, which suggests that the increasing run times are measuring the cost of moving the $M$th and $(M+1)$st pointers. As we increase the range from which quantity is drawn, the mean remains constant, but the tail of the distribution stretches resulting in a noticeable increase in runtimes.

The third test involved measuring the impact of periodic clear events. For this test, we considered only the task of identifying the bids that were involved in exchanges and disentangling them from the other bids in a manner that leaves a valid data structure. Table 3 shows the run time to insert 90,000 bids with clears triggered every 1000, 5000, or 10,000 bids. In the 4-Heap algorithm, clearing is nearly instantaneous because the bids are already organized into disjoint sets. In the Auc-IPR algorithm, clearing is more costly, and thus more frequent clearing slows the algorithm down. By instrumenting the algorithm, we determined that the 4-Heap algorithm spends less than 2% of its time handling the clear action, while the Auc-IPR algorithm spends nearly 15% of its time on the clear. However, the overall better performance of Auc-IPR remains better than 4-Heap.

The final test we ran involved manipulating the order in which bids are submitted to the algorithms. We simulated four extreme scenarios, each with 10,000 bids, alternating between buy offers and sell offers. In the first two scenarios, the buy offers are recieved in decreasing order (B$\searrow$), and the sell offers in increasing order (S$\nearrow$). In the first, all of the buy offers are above

| **Scenario** | **Bid Order** | **Overlap** | **4-HEAP** | **AUC-IPR** |
|---|---|---|---|---|
| 1 | B ↘, S ↗ | $\sim$ 100% | 0.131 | 9.808 |
| 2 | B ↘, S ↗ | $\sim$ 50% | 0.122 | 4.911 |
| 3 | B ↗, S ↘ | 0% | 0.121 | 4.831 |
| 4 | B ↗, S ↘ | $\sim$ 50% | 0.507 | 4.865 |
| 5 | Random | $\sim$ 50% | 0.492 | 0.089 |

Table 4: Time (in seconds) to insert 10,000 bids under different assumptions about ordering.

the sell offers, and in the second, about half of the bids would transact.[5] In Scenarios 3 and 4, the buy offers are recieved in increasing order (B↗), and the sell offers in decreasing order (S↘). In Scenario 3 the bids do not overlap and there are no transactions, while in Scenario 4, about half of the bids would transact.

The first observation is that strictly ordered bids benefits the 4-HEAP algorithm and negatively impacts the performance of the AUC-IPR algorithm. The latter effect is caused by the fact that inserting objects into a tree *in order* is a worst case scenario. For a binary tree, it would result in a tree with a single path. When receiving data in this form, the IPR algorithm is forced to attempt to re-balance the tree with every new bid. On the other hand, ordered data can be beneficial to a heap-based algorithm when the data is presented highest-priority first; the heap algorithm simply inserts new nodes at the bottom of the heap and is never forced to move the nodes higher in the heap.

Scenario 1 seems to pose a worst case for the AUC-IPR algorithm. Not only are the bids received in order, causing a tremendous amount of rebalancing, but many of the bids cause the $M$th and $(M+1)$st pointers to swing from one side of the tree to the other. On the other hand, Scenario 4 is the most difficult for the 4-HEAP algorithm. Although Scenarios 2 and 4 both result in approximately 50% of the bids transacting, in Scenario 2 the transacting bids are received first, while in Scenario 4 they are received last. As new winning bids are received, they bump previously winning bids from the IN heaps into the OUT heaps. In Scenario 2 the OUT heaps are essentially empty, but in Scenario 4 they are full and it is more costly to insert new bids.

The random bid distribution is also presented as a benchmark. The sce-

[5] This statement is purposefully inexact because bid quantities are randomly determined. Thus, even though exactly half of the buy bids have prices greater than exactly half of the sell bids, the actually clearing point is determined by the price that balances the number of units.

nario with random bid values favors the IPR tree algorithm because it will spend relatively little time rebalancing.

# 6 Conclusion

This paper presents an enhanced version of the 4-HEAP algorithm and a novel application of IPR trees to manage bid data in multi-unit double auctions. The AUC-IPR algorithm uses pointers to keep track of the current position of the $M$th and $(M+1)$st units, which enable it to generate price quotes in constant time. However, in order to maintain and efficiently use this information requires some extra bookkeeping.

AUC-IPR has some advantages over 4-HEAP, particularly when the bids are submitted in an unordered form. In the majority of our tests, AUC-IPR proved to be two to three times faster than 4-HEAP. However, when bids are received in order—which could happen if the rules of the auction require bids to improve over the price quote—the 4-HEAP algorithm performed better. This suggests that the selection of an algorithm to use in practice should depend on the expected distribution of bids, which in turn can be affected by the rules.

# Acknowledgments

This research was funded by NSF CAREER award 0092591-0029728000, and the E-Commerce program at NC State. We would like to acknowledge William Walsh for suggesting the multi-unit improvement to the 4-HEAP algorithm, and Arne Andersson for asking the question that motivated the research. We also thank Rada Chirkova and Munindar Singh for their comments on the first author's Masters thesis. The views and conclusions contained herein are those of the authors, as are any errors.

# References

[1] S. Bao. A comparison of two algorithms for clearing multi-unit bid double auctions. Master's thesis, North Carolina State University, 2002.

[2] D. Friedman and J. Rust, editors. *The Double Auction Market: Institutions, Theories, and Evidence*. Addison-Wesley Publishing, Reading, MA, 1993.

[3] G. H. Gonnet. Balancing binary trees by internal path reduction. *Communications of the ACM*, 26(12):1074–1081, December 1983.

[4] K. A. McCabe, S. J. Rassenti, and V. L. Smith. Auction institutional design: Theory and behavior of simultaneous multiple-unit generalizations of the Dutch and English auctions. *American Economic Review*, 80(5):1276–1283, 1990.

[5] D. Pennock, S. Lawrence, C. L. Giles, and F. Nielsen. Extracting collective probabilistic forecasts from web games. In *Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco, CA, 2001.

[6] M. A. Satterthwaite and S. R. Williams. Bilateral trade with the sealed bid $k$-double auction: Existence and efficiency. *Journal of Economic Theory*, 48:107–133, 1989.

[7] A. L. Tharp. *File Organization and Processing*. John Wiley & Sons, 1988.

[8] W. Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16:8–37, 1961.

[9] W. E. Walsh. A note about the 4-heap algorithm. Private communication, 2000.

[10] P. R. Wurman, W. E. Walsh, and M. P. Wellman. Flexible double auctions for electronic commerce: Theory and implementation. *Decision Support Systems*, 24:17–27, 1998.

[11] P. R. Wurman, M. P. Wellman, and W. E. Walsh. A parametrization of the auction design space. *Games and Economic Behavior*, 35(1-2):304–338, April-May 2001.