IBM Research Report

Efficient Winner Determination Techniques for Internet Single Item Multi-Unit Open-Cry Auctions

Achal Bassambo[1], **Manish Gupta**[2], **Sandeep Juneja**[3]
email: gmanish@in.ibm.com

IBM Research Division
IBM India Research Lab
Block I, I.I.T. Campus, Hauz Khas
New Delhi - 110016. INDIA.
Phone: +91-11-6861100, Fax: +91-11-6861555

IBM Research Division
Almaden - Austin - Beijing - Delhi - Haifa - T.J. Watson - Tokyo - Zurich

[1]achalb@stanford.edu, Graduate School of Business, Stanford University
[2]gmanish@in.ibm.com, IBM India Research Laboratory, Indian Institute of Technology, New Delhi
[3]sandeepj@cse.iitd.ernet.in, Indian Institute of Technology, New Delhi

Efficient Winner Determination Techniques for Internet Single Item Multi-Unit Open-Cry Auctions

IBM India Research Lab
Block I, Indian Institute of Technology
Hauz Khas, New Delhi - 110016. India.
email: gmanish@in.ibm.com

Abstract

Internet auction servers have a history of breaking down when a large number of accesses are made by users simultaneously, i.e., under peak load. In this paper we develop algorithms for efficiently processing bids in a single item, N unit ($N \geq 1$) open cry auction for a large class of winner determination rules. Existing techniques consider all previously submitted bids along with the new arrivals to update the current set of winners. We propose that at most N 'potential winner bids' amongst the previously submitted bids need to be used for such updates, thus significantly reducing the computation time and memory requirements. This is crucial when a large number of auctions are being conducted simultaneously. For a commonly used greedy auction rule we show that the expected number of potential winner bids may be much less than N under reasonable probabilistic assumptions. Another advantage of our approach vis-à-vis existing approaches is that it is amenable to distributed bid processing.

1 Introduction

The phenomenon of a website attracting millions of users in a short duration is becoming increasingly frequent. On July 14, 1998, the Guinness Book of World Records recognized the 1998 Olympic Games Web Site for two world records [6]. The first record was for receiving 634.7 million requests over the 16 days of the Olympic games. The second record was set when 110,414 hits were received in a single minute around the time of the Women's Figure Skating. We have come a long way since then. The use of Internet to facilitate commerce, and in particular auctions, has also been growing at a rapid rate. It is not unreasonable to expect that an auction sale of a superstar's memorabilia in a sports event may attract thousands of bidders. Such a large number of participants in an auction can put immense load at a web server. For example, eBay Inc.'s auction site collapsed due to heavy load in June 1999 (see [7]). In this paper, we address the issue of speeding up bid processing in a scalable manner in Internet open-cry auctions.

A large number of e-commerce sites have been developed and launched with little or no planning [12]. As e-commerce companies such as eBay, Charles Schwab & Co. and E-Trade Group are discovering the hard way, a key strategy for successfully expanding business on the Web is scaling the computer systems up to keep pace with the rapid growth. The cost of stumble is a blizzard of criticism, millions of dollars lost in expenses and millions more forfeited in customer goodwill. Typically, companies react to growing e-commerce and transactions business (and hence the Internet traffic) by adding more machines. However, many times a cost effective strategy may be to improve efficiency of the algorithms and software that runs on these machines.

Internet commerce technologies for different kinds of auctions: open-cry, sealed bid, and Dutch auction (see [9], [10]) have been developed for Websphere Commerce Suite (WCS 4.1), IBM's commerce server.

While [9] and [10] describe how these auctions may be conducted on the Internet, they do not discuss issues related to improving performance and scalability of these auctions.

In this paper our focus is on quickly determining winner bids in an open-cry auction for N units ($N \geq 1$) of a single item. For such auctions, an obvious allocation strategy is to assign the units to the bidders in a descending order of the price/unit offered by them. However, this may not be feasible many times. For example, a bid desiring multiple units that wants 'all or nothing' may be rejected if the quantity demanded is not available. Also, the same bid may be accepted at a later time (and a previously accepted bid rejected) if a newly arriving bid makes it attractive and feasible to include it along with the rejected bid in the list of 'current winners'. Due to such complications, the existing packages such as WCS 4.1, at any time during an ongoing auction, combine the newly arrived bids with all the ones that had arrived earlier to determine the current set of winners based on the auction rules. In such cases, if the number of bids is large, the processing delays may be large and the server may even breakdown. We develop a method that, for a large class of auction rules, at any time requires at most N well chosen "potential winner bids" (defined in Section 3) to be stored amongst all the bids that have arrived thus far. The remaining bids can be rejected as "loser bids", i.e., they have no chance to qualify as winner bids in future. This considerably reduces the response times to the bidders and the computer storage requirements since this reduced list of potential winner bids can be maintained as a set of persistent objects in the main memory for quick access and modification. An additional benefit of this is that the loser bids get an opportunity to quickly revise their bids. We also discuss how the set of potential winner bids can be efficiently updated when the new bids arrive.

We focus on two commonly used rules for selecting winners: namely, the greedy rule and the knapsack rule (explained in Section 2). The greedy rule is the most popular rule due to the simplicity both in explaining it to the lay person and in its implementation. We develop algorithms that quickly update the list of potential winner bids ($O(\log N)$[1] per bid) when the greedy rules are used. Through simulations we show that this updation procedure provides a drastic improvement over computational effort required by existing packages. We also show that under greedy rules and under some reasonable assumptions on the probability distribution of the quantity required by each bid, the expected number of potential winner bids is much less than N. We outline an algorithm for updating the list of potential winner bids under the knapsack rules. Our simulations show that under knapsack rules as well the average number of potential winner bids is much smaller than N.

There exists some literature related to the online knapsack problem (see [11], [8]). Their focus is somewhat different from ours as they consider auctions where the final decision on whether a bid is a winner or a loser is made as soon as an arriving bid is analysed. This benefit of speedier decision making is at a cost that the solutions obtained may be sub-optimal. We, on the other hand, decide whether a bid is a potential winner bid or a loser bid at the time of its arrival. Our solution is always optimal.

As mentioned earlier, the implementation in WCS 4.1 requires that all bids that have arrived at any given time be considered in determining the set of winners at that time. This centralized procedure makes it difficult to efficiently utilize parallel computing. In this paper, we briefly discuss how the concept of maintaining a list of potential winner bids can be used effectively in a distributed setting. We propose a distributed structure where each processor recieves a set of bids from which it determines the set of potential winner bids. This set of potential winner bids from each processor is then transferred to a single processor that determines the set of current winners (that may be displayed on the web to potential bidders).

In our analysis we have assumed that the bidder cannot retract a bid once the bid has been made.

[1] A function $f(N)$ is said to be $O(g(N))$ if there exists a constant $K > 0$ so that $f(N) \leq Kg(N)$ for all N sufficiently large. It is said to be $\Theta(g(N))$ if there exist constants $0 < K_1 < K_2$ so that $K_1 g(N) < f(N) \leq K_2 g(N)$ for all N sufficiently large

In case retraction is allowed and a potential winner bid is retracted, the new set of potential winners may be determined by considering all the bids that have arrived at the auction (including the previously declared loser bids).

In Section 2 we review the commonly used winner determination auction rules for open cry auctions. In particular, we review the existing implementation of WCS 4.1. In Section 3 we state our assumptions on the auction rules that we consider and the theorem identifying the potential winner bids under these assumptions. The potential winner bids set updation techniques under the greedy rule are discussed in Section 4. This section also states a theorem on the order of magnitude of the expected number of potential winner bids for large values of N. The updation techniques under knapsack rules are briefly discussed in Section 5. In Section 6 we discuss the results of the simulations conducted to validate our conclusions. We discuss how the key ideas of the paper can be implemented in a distributed setting, in Section 7. In Section 8, we briefly discuss some issues related to processing *order bids*, i.e., a software agent that bids on behalf of a human bidder.

2 Open-Cry Auction: An Overview

We refer the reader to [9] and [10] for a general introduction to auctions on Internet and to issues specific to Internet auctions. In this section we briefly describe the open cry auction, the greedy and knapsack auction rules and the implementation of WCS 4.1.

In an open-cry auction (also popularly known as "English" auction) of a single item with multiple units, the seller may specify the minimum starting bid. A bidder may also specify a minimum bid increment that a new bid needs to have over the current best price (based on bids of current winners) to be eligible [2]. All currently eligible bids are displayed to the users. The set of current winner bids (based on pre-specified auction rules) is also identified. At the time of bidding, bidder specifies the number of units desired. A bidder may also specify whether he would accept partial quantity or not. Once the bidding phase is over, the bidders with the high bids get the items being auctioned, but the price they pay could be different than what they bid. In a discriminative auction, also known as a yankee auction, the winners pay the amount that they bid. In a non discriminative auction the winning bidders pay the price paid by the winning bidder with the lowest bid (this is currently the trend on the Internet; sites like www.ebay.com use this methodology for auctioning off multiple items).

We now state some auction rules that currently exist at auction sites on the Internet. The following rules (taken from http://pages.ebay.com/help/buyerguide/bidding-type.html) are for the "Dutch" auction format used by a seller for selling multiple copies of an item on eBay.com:

- Sellers start by listing a minimum price, or starting bid, and the number of (identical) items for sale.
- Bidders specify both a bid price and the quantity they want to buy.
- All winning bidders pay the same price; which is the lowest successful bid. That might be less than what you bid!
- If there are more buyers than items, the earliest successful bids get the goods.
- Higher bidders get the quantities they've asked for.
- Bidders can refuse partial quantities.

[2] A bid is considered eligible if at the time of submission it passes all the checks such as check on bidder's creditworthiness, minimum bid criteria, etc.

The above is simply an open-cry auction having a starting price but no bid increment, and non discriminative pricing. Amazon.com and other sites too have the same auction format for selling multiple quantities of an item. We define the rules similar to those used in the Dutch auction at eBay.com as *greedy* type.

2.1 Greedy Rules for Winner Determination

Some notation is needed before we describe the greedy rules for winner determination. Recall that N denotes the number of units on a single item multi-unit open-cry auction. For any bid b let $val(b)$ denote the price per unit offered. Let $Q(b)$ denote the quantity requested in the bid and let $T(b)$ denote the time of arrival of the bid. Note that if a bid b is willing to accept partial assignment of the quantity bid, then from winner determination viewpoint we may treat b as comprising $Q(b)$ identical bids, each bidding for a single unit. This allows us to assume without loss of generality that all bids in the auction do not accept partial assignments ,i.e, **they want all or nothing**. Let R denote a set of rules of an open cry auction. For example, we set $R = greedy$, when the greedy method of winner determination is used and set it to *knapsack* when the knapsack rules are used. Let $W(R, Z, N)$ denote the winner bids amongst bids Z in an auction for N units of a single item, where winners are selected using rules R.

Given a set of bids Z, the greedy rules for winner determination would typically sort all the bids so that given any two bids b_i and b_j, the sorting places b_i above b_j iff:

- $val(b_i) > val(b_j)$ or,
- $val(b_i) = val(b_j)$ and $Q(b_i) > Q(b_j)$, or,
- $val(b_i) = val(b_j)$, $Q(b_i) = Q(b_j)$ and $T(b_i) < T(b_j)$.

We use the convention $b_i > b_j$ ($b_i < b_j$) to mean that bid b_i is placed above (below) bid b_j in the sorted list. In case $b_i = b_j$ one may break the tie arbitrarily by assuming small perturbations in the time of bid arrival. We refer to this method of sorting as **gsort**.

Additional notation is needed to facilitate further discussion. Let n denote the cardinality of the set Z. Let $(b_1, b_2, \ldots, b_n)$ denote the list of bids in Z sorted in descending order using gsort. The algorithm for determining $W(greedy, Z, N)$, call it **A1**, relies on the observation that under greedy rules, if $Q(b_1) > N$ then b_1 is rejected so that $W(greedy, Z, N) = W(greedy, Z - \{b_1\}, N)$. Otherwise it is accepted and $W(greedy, Z, N) = b_1 \cup W(greedy, Z - \{b_1\}, N - Q(b_1))$. Variables $(X_i : i \leq n)$ and $\mathcal{Z}$ are defined in the algorithm and will be useful in our analysis later.

2.2 The algorithm A1

$$
\begin{aligned}
\text{Set } \mathcal{Z} &= \emptyset, X_1 = N, j = 1; && (1)\\
\text{while}(Z &\neq \emptyset \text{ or } X_j \neq 0)\{ && (2)\\
\text{if}(Q(b_j) &> X_j) \text{ set } Z = Z - \{b_j\}, X_{j+1} = X_j; && (3)\\
\text{else set } \mathcal{Z} &= \mathcal{Z} \cup b_j, X_{j+1} = X_j - Q(b_j); && (4)\\
j &= j + 1;\} && (5)\\
\text{return } &\mathcal{Z}; && (6)
\end{aligned}
$$

The set $\mathcal{Z}$ returned from this algorithm denotes the set $W(greedy, Z, N)$. Note that the algorithm traverses the sorted list of bids from top to bottom to allocate the auction quantity to the bids one at a time. For all j, after bids $\{b_1, , \ldots, b_{j-1}\}$ have been examined, it proceeds recursively to solve the smaller

problem of determining $W(greedy, Z^j, X_j)$, where $Z^j = \{b_j, \ldots, b_n\}$. We see later that a set of potential winners can be obtained by a slight modification of the above algorithm.

In practice, some variants of greedy rules may sort in a slightly different manner from gsort (e.g., if the two bids quote the same value per unit, then one may be selected over the other at random). To keep things simple we focus on the greedy rule that sorts using gsort and selects winners as described by **A1**. As will become apparent to the reader, the key ideas of this paper can be adjusted to suit variants of the greedy rule through simple modifications.

2.3 Knapsack rules for Winner Determination

The winners are chosen by solving an integer knapsack problem (see, e.g., [5]) so that the revenue to the seller is maximized. To see this precisely, consider again the set $Z = (b_1, b_2, \ldots, b_n)$. Let $(I_i^* : i = 1, \ldots, n)$ be a solution to the following integer program:

$$\max \sum_{i=1}^{n} I_i * val(b_i) * Q(b_i)$$

$$\text{s. t.} \sum_{i=1}^{n} I_i * Q(b_i) \leq N, \text{ and}$$

$$(I_i : i = 1, \ldots, n) \text{ equals 1 or 0.}$$

Then $W(knapsack, Z, N)$ consists of each bid b_i for which $I_i^* = 1$. Since the winner bids obtained using the greedy rule form a feasible solution to this problem, the seller's revenue under greedy rule is less than or equal to his revenue under knapsack rule. However, it is easily seen that if all the bids either request a single item or are partial bids, i.e., bidders are ready to accept any quantity less than what is requested in their bids, then the greedy solution is equivalent to the the knapsack solution.

2.4 Implementation of Winner Determination in WCS 4.1

Assume that the auction starts at time $t = 0$ and continues till time $t = T_{end}$. For $0 \leq t_1 \leq t_2 \leq T_{end}$, let $S(t_1, t_2)$ denote the set of bids that arrive after time t_1 and before or at time t_2. Let $S(t)$ denote $S(0, t)$. The winner determination algorithm in WCS 4.1 works as follows: All arriving bids are stored in a database. A process that computes winner bids runs periodically every τ seconds (currently τ is kept at 90 seconds). During run $k \geq 1$ of this process (at time $k\tau$), all the bids in the database are considered, i.e., the set of bids $S((k-1)\tau)$ that were present at the time of the last run, as well as the set of bids $S((k-1)\tau, k\tau)$ that arrived after the last run. The greedy rules are used to determine the new set of winners from the resulting set $S(k\tau) = S((k-1)\tau) \cup S((k-1)\tau, k\tau)$.

To get an approximate idea of the computational effort involved, let m denote the cardinality of the set $S((k-1)\tau, k\tau)$ and l denotes the cardinality of the set $S((k-1)\tau)$. Clearly, in the above algorithm, the computational effort for sorting the newly arrived m bids along with the existing (already sorted) l bids is $O(m \log(m + l))$ (see [3]). Recall from Section 2.1 that to determine the winners we need to traverse the sorted list from the top to the bottom. As we consider each bid one by one, we declare a bid a winner if the quantity desired in the bid can be satisfied by the remaining quantity on hand. We continue allocating the auction quantity to bids until either we allocate all the auction quantity or we reach the bottom of the list. Therefore, the additional computational effort in finding the current winners is in the worst case $O(m + l)$. During peak load, m may be quite large (much larger than N) and thus large amount of computational effort may be required. In the next section we present algorithms that drastically reduce the computational effort.

3 Identifying the Set of Potential Winners

In this section first we make a set of assumptions that impose minimal restrictions and cover a large class of practically implemented rules for open-cry auction. Then we show that under these assumptions, at any time t, at most N bids need to be retained from the set $S(t)$ to determine winners of the auction at any time in future and the rest can be discarded as loser bids. Some notation is needed for this purpose.

Let $S^*(t)$ denote the bids $S(t, T_{end})$. Let $S_q(t) \subset S(t)$ denote the set of all bids in $S(t)$ asking for quantity q for $q \leq N$. For notational simplicity we suppress the reference to t and let S, S^*, S_q denote $S(t), S^*(t), S_q(t)$ whenever this does not cause confusion. It is worth keeping in mind that the results shown below involving S and S^* hold for *any* two disjoint sets S and S^*.

Assumption 1 *Under the set of rules R, there exists a sorting criteria $\mathcal{C}$ for the bids in S_q ($q \leq N$) such that any bid in the list sorted using $\mathcal{C}$ belongs to the winner set $W(R, S, N)$ only if all the bids above it in the list belong to $W(R, S, N)$.*

Note that this assumption is satisfied by the greedy and the knapsack rules where the sorting is based on gsort.

For any set Z, let $\#(Z)$ denote its cardinality. For any number x, let $\lfloor x \rfloor$ denote the greatest integer less than or equal to x. When Assumption 1 holds, and if $\#(S_q) \geq \lfloor N/q \rfloor$, then let $T_q(S_q)$ denote the set of top $\lfloor N/q \rfloor$ bids of the set S_q under $\mathcal{C}$. Otherwise, if $\#(S_q) < \lfloor N/q \rfloor$ then let $T_q(S_q) = S_q$. Let $T(S) = \cup_{q=1}^{N} T_q(S_q)$. The following proposition easily follows from Assumption 1:

Proposition 1 *Under Assumption 1, $W(R, S \cup S^*, N) = W(R, T(S) \cup S^*, N)$.*

This result implies that in any ongoing auction, if Assumption 1 holds, then all potential winner bids from set S lie in the subset $T(S)$. In particular, the bids in the set $S - T(S)$ clearly are the loser bids and can be rejected.

Lemma 1 *The following relation holds:*

$$\#(T(S)) \leq N(1 + \log N).$$

Proof: Note that

$$\#(T(S)) = \sum_{q=1}^{N} \#(T(S_q)) \leq \sum_{q=1}^{N} N/q \leq N(1 + \sum_{q=2}^{N} 1/q).$$

The result follows by noting that $1/q \leq \int_{q-1}^{q} (1/x)dx$. $\Box$.

Thus, under Assumption 1, $O(N \log N)$ bids need to be kept at any time and the rest can be discarded. We now show that the number of potential winners can be further reduced under the following two assumptions:

Assumption 2 *Under the set of rules R, for all N, if $b \in Z$ and $b \notin W(R, Z, N)$, then $W(R, Z, N) = W(R, Z - \{b\}, N)$.*

Thus, under this assumption, the removal of any loser bid from the set of bids Z does not alter the set of winner bids.

Assumption 3 *Under the set of rules R, for all N, if $b \in W(R, Z, N)$ then:*

$$W(R, Z - \{b\}, N - Q(b)) = W(R, Z, N) - \{b\}.$$

Thus, if one winner bid b is assigned $Q(b)$ units of the item, the other winners (call them B) are not assigned any unit and the remaining $N - Q(b)$ units are re-auctioned to the $Z - \{b\}$ bids, then the winners of the re-auction are the other winners of the previous auction, i.e., the set B. Many practically concievable rules for the open-cry auction satisfy Assumptions 2 and 3. In particular, the greedy and the knapsack rules do satisfy these assumptions.

For notational brevity let $G(R, S, N)$ denote the set $\cup_{q=1}^{N} W(R, S, q)$, let Z_1 denote the set $W(R, S \cup S^*, N) \cap S$ and let Z_2 denote the set $W(R, S \cup S^*, N) \cap S^*$. Thus, Z_1 and Z_2 denote the global winners that came from the sets S and S^*, respectively. Now we state our main result.

Theorem 1 *If R satisfies Assumptions 2 and 3, then*

$$Z_1 \subset G(R, S, N). \tag{7}$$

In addition, for all $j \geq 1$,

$$\#(G(R, S, j)) \leq \#(G(R, S, j-1)) + 1 \leq j. \tag{8}$$

Thus, the above theorem states that if Assumptions 2 and 3 hold then at any time t, atmost N bids amongst S are potential winners. In particular, the bids in $S - G(R, S, N)$ are the loser bids that can be rejected. We shall henceforth refer to $G(R, S, N)$ as the set of *potential winner bids.*

Proof of Theorem 1: Assumption 2 implies that

$$Z_1 \cup Z_2 = W(R, S \cup Z_2, N).$$

Assumption 3 further implies that

$$Z_1 = W(R, S, N - \sum_{b \in Z_2} Q(b)),$$

from which (7) follows. We prove (8) using induction on the number of identical items on auction. Note that $\#(G(R, S, 1))$ equals 0 or 1, depending on whether $W(R, S, 1)$ is empty or not. Assume that (8) holds for all $j \leq m$. We now show that it holds for $m + 1$ by assuming the contrary and showing a contradiction. Thus, suppose that there exist bids $b_1, b_2 \in G(R, S, m+1)$ and $b_1, b_2 \notin G(R, S, m)$. This implies that $b_1, b_2 \in W(R, S, m+1)$. From Assumption 3, we have that $W(R, S - \{b_1\}, m + 1 - Q(b_1)) = W(R, S, m+1) - \{b_1\}$. Since $W(R, S - \{b_1\}, m + 1 - Q(b_1)) \subset G(R, S, m)$, it follows that $b_2 \in G(R, S, m)$ giving us the desired contradiction. $\Box$.

Note that it is easy to construct an S^* so that Z_2 requires any quantity from 0 to N. Therefore any set $(W(R, S, i) : 1 \leq i \leq N)$ can be a subset of a winner set for an appropriate S^*. In particular, for any bid $b \in G(R, S, N)$, there exists an S^* so that b is a final winner bid. It follows that the set of potential winners cannot be further trimmed.

3.1 Updating the potential winners set

Assume that given a set of bids Z we have an algorithm for determining $G(R, Z, N)$ and $W(R, Z, N)$. Also suppose that we wish to compute the current set of winners at discrete time intervals $(t_1, t_2, \ldots, t_k)$, for some $k \geq 1$. Then the above theorem suggests the following procedure for updating the current set of winners:

At time t_1 the set of potential winners $G(R, S(t_1), N)$ is determined and retained and the rest of the bids $S(t_1) - G(R, S(t_1), N)$ are discarded. Proceeding inductively, assume that at any time t_i the set $G(R, S(t_i), N)$ is retained and the rest are discarded. Note that due to Assumption 2

$$W(R, S(t_i), N) = W(R, G(R, S(t_i), N), N),$$

therefore only bids $G(R, S(t_i), N)$ are needed to compute the current set of winners at time t_i. Furthermore, due to Theorem 1,

$$W(R, S(t_{i+1}), j) = W(R, G(R, S(t_i), N) \cup S(t_i, t_{i+1}), j)$$

for $1 \leq j \leq N$. Taking the union for $1 \leq j \leq N$, we get

$$G(R, S(t_{i+1}), N) = G(R, G(R, S(t_i), N) \cup S(t_i, t_{i+1}), N).$$

Hence, the newly arriving bids $S(t_i, t_{i+1})$ may be combined with $G(R, S(t_i), N)$ to determine $G(R, S(t_{i+1}), N)$ at time t_{i+1}. The bids $G(R, S(t_i), N) \cup S(t_i, t_{i+1}) - G(R, S(t_{i+1}), N)$ are then discarded. The future updation may continue in a similar manner.

4 Potential Winners Set Updation under Greedy Rules

We now discuss how the set of potential winner bids is determined and updated when the new bids arrive. To this end, Lemma 2 states the key recursive relationship. To aid in its statement let $r(x, y)$ denote $\max(x - y, y - 1)$, let n denote the cardinality of of the set of bids Z and let $(b_1, b_2, \ldots, b_n)$ denote the list of these bids sorted in descending order using gsort.

Lemma 2 *For $j \geq Q(b_1)$,*

$$G(greedy, Z, j) = b_1 \cup G(greedy, Z - \{b_1\}, r(j, Q(b_1)). \tag{9}$$

For $j < Q(b_1)$,

$$G(greedy, Z, j) = G(greedy, Z - \{b_1\}, j). \tag{10}$$

Proof: Note that (10) straightforward, hence we prove (9). For $i < Q(b_1)$, it is obvious that:

$$W(greedy, Z, i) = W(greedy, Z - \{b_1\}, i). \tag{11}$$

From greedy rules it is clear that for $i = Q(b_1), \ldots, j$:

$$W(greedy, Z, i) = b_1 \cup W(greedy, Z - \{b_1\}, i - Q(b_1)). \tag{12}$$

Hence, from (11) and (12), it follows that

$$G(greedy, Z, j) = (\cup_{i=1}^{Q(b_1)-1} W(greedy, Z - \{b_1\}, i)) \cup b_1 \cup (\cup_{k=1}^{j-Q(b_1)} W(greedy, Z - \{b_1\}, k))$$

and (9) follows. □.

In view of Lemma 2 the algorithm for determining $G(greedy, Z, N)$ is a simple modification of **A1**. This algorithm, call it **A2**, is identical to **A1** except that in (4), the step $X_{j+1} = X_j - Q(b_j)$ is replaced by $X_{j+1} = r(X_j, Q(b_j))$.

Note that the output $\mathcal{Z}$ is the set $G(greedy, Z, N)$ and that the elements in $\mathcal{Z}$ may easily be maintained as a sorted list (based on gsort). Also, analogous to **A1**, for all j, after bids $\{b_1, , \ldots, b_{j-1}\}$ have been examined, **A2** proceeds recursively to solve the smaller problem of determining $G(greedy, Z^j, X_j)$, where $Z^j = \{b_j, \ldots, b_n\}$. If the above algorithm terminates because $X_j's$ decrease to zero, we say that $G(greedy, S, N)$ is *full*.

The computational effort required in the above algorithm is $O(n \log n)$ for sorting Z and $O(n)$ for finding $G(greedy, Z, N)$. For large set Z the effort required in sorting it can be high. We now discuss some updation procedures that do not require the sorted Z.

4.1 Adding bids one at a time

At the start of the auction the set $\mathcal{Z}$ is empty. Suppose that at any time t it maintains a sorted list of bids (using gsort) in $G(greedy, S(t), N)$. We update $\mathcal{Z}$ after each arriving bid. Let b_{new} denote the newly arrived bid after time t. This can be inserted in $\mathcal{Z}$ in its appropriate sorted position in $O(\log N)$ steps (using binary sort techniques). The algorithm **A2** can then be used to find $G(greedy, S(t) + \{b_{new}\}, N)$ in $O(N)$ steps. The output of the algorithm gives the updated $\mathcal{Z}$. All bids in $G(greedy, S(t) + \{b_{new}\}, N)$ -$G(greedy, S(t), N)$ are rejected. Thus if there are a total of n bids introduced one at a time, this procedure requires $O(nN)$ computational effort or $O(N)$ computational effort per bid.

4.2 Adding k bids at a time

Rather than updating $\mathcal{Z}$ by adding one bid at a time, it may be computationally efficient to collect a batch of $k > 1$ bids and then update $\mathcal{Z}$. To determine the order of magnitude of the computational effort involved note that adding k bids at the correct sorted position in the set $\mathcal{Z}$ requires $O(k \log(k + N))$ operations. If Z_k denotes the bids in the batch of size k, then computing $G(greedy, \mathcal{Z} \cup Z_k, N)$ requires $O(k + N)$ operations. Thus, per bid the number of operations required is $O(\log(k + N)) + O(1 + N/k)$. It follows that for $k = O(N)$, per bid computational effort is $O(\log N)$, significantly better than when $k = O(1)$. This conclusion is verified using simulation in Section 6.

4.3 An initial acceptance test for arriving bids

From practical viewpoint, if, after announcing the current set of winners, an auction recieves a large number of bids (much larger than N), then there is a significant chance that an arrived bid will not change $\mathcal{Z}$. Fortunately, this determination can be made in an $O(1)$ time. For this purpose we define the arrays $((\hat{V}(q), \hat{Q}(q)) : q \leq N)$. These arrays are updated in **A2** in (4) where we set $\hat{V}(X_j - i) = val(b_j)$ and $\hat{Q}(X_j - i) = Q(b_j)$ for $i = 0, 1, \ldots, X_j - X_{j+1} - 1$. Thus, whenever $X_{j+1} < X_j$, the value and the size of the bid that cause the reduction are recorded. Note that computing these arrays requires extra $\Theta(N)$ time in **A2**.

The following lemma explains how the array $((\hat{V}(q), \hat{Q}(q)) : q \leq N)$ helps in deciding whether the newly arrived bid belongs to the set of potential winners. Its proof is given in the appendix.

Lemma 3 *Suppose that* $((\hat{V}(q), \hat{Q}(q)) : q \leq N)$ *are determined using* **A2**, *then* $b_{new} \in G(greedy, Z + \{b_{new}\}, N)$ *if and only if either*

$$val(b_{new}) > \hat{V}(Q(b_{new}))$$

or

$$val(b_{new}) = \hat{V}(Q(b_{new}))$$

and $Q(b_{new}) > \hat{Q}(Q(b_{new}))$.

We call the comparisons in the above lemma as the initial acceptance test. When the bids are added one at a time to $\mathcal{Z}$, every time b_{new} is added to the list of potential winners, the arrays $((\hat{V}(q), \hat{Q}(q)) : q \leq N)$ are updated in $O(N)$ time. Thus, if p denotes the fraction of new bids rejected due to the initial acceptance test then clearly the total computational effort of incorporating n bids is of order $O(pn + (1 - p)nN)$.

Similarly, when a batch of $\Theta(N)$ arriving bids is used to update $\mathcal{Z}$, then the computational effort is of order $O(pn + (1 - p)n \log N)$. However, a drawback of having a large batch is that the probability of rejecting a new bid due to the initial test now reduces as the complete batch sees the same variables $((\hat{V}(q), \hat{Q}(q)) : q \leq N)$. The benefits of the initial acceptance test are verified through simulation in Section 6.

4.4 Expected number of potential winner bids

Although, we have an upper bound of N on $G(greedy, Z, N)$ and it is easy to create cases where this bound is achieved (e.g., when all the top bids request a single item), simulation experiments show that number of bids in $G(greedy, Z, N)$ is typically much smaller. We now conduct an order of magnitude analysis of the expected number of bids in $G(greedy, Z, N)$ under some reasonable assumptions on the probability distribution of bid sizes. Our conclusion is that for large N, this expectation typically is much smaller than N.

Let τ denote the number of bids in the set $G(greedy, Z, N)$. Recall from the definition of the sequence $(X_i : i \geq 1)$ that every time a bid is added to the set $\mathcal{Z}$, the value of the appropriate X_i decreases. This motivates the definitions of following stopping times (e.g., see [4] as a reference on stopping times): Let $T_0 = 1$ and $T_i = \min\{j : X_j < X_{T_{i-1}}\}$ for $i \leq \tau$. Thus, b_{T_i} is the i th bid entering $\mathcal{Z}$ in **A2**. Let $Y_i = X_{T_i}$ for all $0 \leq i \leq \tau$. Thus Y_i records the i th unique value of the sequence $(X_i : i \leq T_\tau)$. It follows that:

$$\begin{aligned} Y_{i+1} &= r(Y_i, Q(b_{T_i})) && (13) \\ &= \max(Y_i - Q(b_{T_i}), Q(b_{T_i}) - 1). && (14) \end{aligned}$$

Assumption 4 explains the class of probability distributions on the sizes of bids, that we consider. This assumption is somewhat restrictive to avoid undue mathematical technicalities. It however covers many cases of practical relevance (as discussed later) and provides insight explaining the order of magnitude of the expected number of bids in $G(greedy, Z, N)$.

Assumption 4 *There exist positive numbers $\bar{Q}, \beta, \gamma$ and $0 \leq \alpha \leq 1$ such that $\bar{Q} = \Theta(N^\alpha)$, and*

$$\beta/N^\alpha \leq P(Q(b_i) = q) \leq \gamma/N^\alpha$$

for $q \leq \bar{Q}$ for all $i \leq n$ and $P(Q(b_i) = q) = 0$ otherwise for all $i \leq n$.

For example, the case where each $Q(b_i)$ takes values from 1 to N with equal probability is modelled by taking $\alpha = 1$, $\bar{Q} = N$ and $\beta = \gamma = 1$. To capture the setting where the bids are predominantly small in size, we may consider a smaller $\bar{Q}$, for example, $\bar{Q} = \Theta(\sqrt{N})$ or $\bar{Q} = \Theta(1)$. To keep things simple we further assume that n (i.e., the cardinality of Z) is sufficently large so that $Y_\tau = 0$ (i.e., $G(greedy, Z, N)$ is full). This assumption only increases the value of τ and hence the bound on $E(\tau)$ under this assumptions also holds without the assumption.

Before we state the theorem additional notation is needed. Let $\tau_1 = \min\{i \geq 1 : Y_i \leq 2\bar{Q}\}$. Let $\tau_2 = \tau - \tau_1$.

Theorem 2 *Under Assumption 4, $E(\tau) = \Theta(N^{1-\alpha}) + O(\log N)$.*

The following two lemmas prove this theorem. Their proofs are given in the appendix.

Lemma 4 *Under Assumption 4*

$$E(\tau_1) = \Theta(N^{1-\alpha}). \quad (15)$$

Lemma 5 *Under Assumption 4, $E(\tau_2) = O(\log N)$*

Thus, if the quantity requested by bids is large, for example if $\alpha = 1$ ($\bar{Q} = \Theta(N)$) then the number of potential winner bids is small $O(\log N)$. On the other hand if most of the bids are small, for example if $\alpha = 0$ and $\bar{Q} = \Theta(1)$ then the number of potential winner bids is large $\Theta(N)$.

5 Updation of Set of Potential Winners under Knapsack Rule

There exists extensive literature dealing with methods for solving the integer knapsack problem (see, e.g., [2]). In this section we provide a brief discussion on how $G(knapsack, S, N)$ may be updated using simple extensions of well known dynamic programming recursions. We do not focus on developing efficient updation techniques, a topic that needs further investigation.

For notational brevity, let $W_i(t)$ denote the set of bids $W(knapsack, S(t), i)$. Let $V(U)$ denote the total value of all the bids in the set U. Now at any time t, we maintain a separate list of each $W_i(t)$ for $i \leq N$. Recall that $G(knapsack, S(t), N) = \cup_{i=1}^{N} W_i(t)$. We also keep track of $(V(W_i(t)) : 1 \leq i \leq N)$. Also, for each bid $b \in \cup_{i=1}^{N} W_i(t)$, we maintain a set $\mathcal{L}_b = (i : b \in W_i)$.

Then a newly arrived bid b_{new} updates this data structure as follows: For each $i \geq Q(b_{new})$, if

$$V(W_{i-Q(b_{new})}) + val(b_{new}) * Q(b_{new}) > V(W_i), \tag{16}$$

then W_i is updated to include b_{new}. Let W_i' denote the updated structure. The $\mathcal{L}_{b_{new}}$ is updated to include i. All bids $b \in W_{i-Q(b_{new})}, b \notin W_i$ are also included in W_i' and their $\mathcal{L}_b$ are updated to include i. For all bids $b \in W_i, b \notin W_i'$, the corresponding $\mathcal{L}_b$ is updated to exclude i. $V(W_i')$ is set to $V(W_{i-Q(b_{new})}) + val(b_{new}) * Q(b_{new})$. Otherwise if (16) does not hold then W_i remains unchanged. After this update is completed for each $i \geq Q(b_{new})$, all updated W_i' replace the W_i. Each b such that $\mathcal{L}_b = \emptyset$ is rejected. This process of updation continues one bid at a time.

6 Simulation Results

In this section we use simulation to validate our conclusions. We first consider the updation methods under the greedy auction rules proposed in Section 4. We show that it is computationally efficient to update the potential winners set using batches of size greater than 1. In particular, batch size close to N performs quite well. We also show that incorporating the initial acceptance test (discussed in Section 4.3) further significantly reduces the computational effort. Next we show that the proposed algorithm using appropriate batch size and the initial acceptance test is much faster than the naive WCS 4.1 implementation. Finally, we show that under some assumptions on the probability distribution of the bid sizes, the average number of bids in the potential winners set is much smaller than N both under the greedy auction rule and under the knapsack auction rule. For the greedy case, this verifies the results of Theorem 2.

Our simulations were performed on a pentium machine with windows NT operating system. In all our simulations we assume that the bid values are independent of the bid sizes. The distribution of the bid sizes is taken to be uniform between 1 and the maximum bid size (specified later for each experiment). Note that if the bid values come from a continuous distribution, the probability of having ties is zero. Thus, gsort places b_i above b_j with positive probability iff $val(b_i) > val(b_j)$. Note that for any collection of i.i.d. random variables $(A_i : 1 \leq i \leq n)$ from a continuous probability distribution the probabilities of observing a particular ordering is independent of their probability distribution (e.g., $P(A_1 \geq A_2 \geq \ldots \geq A_n) = 1/n!$ irrespective of the distribution of A_i). Thus under greedy rules, since gsort is used to sort bids, the relative position of the bids (and hence the winner list) is independent of the probability distribution of the bid values if the bid value has a continuous probability distribution. For simplicity, we took the distribution of the bid values to be uniformly distributed, taking values between 0 and 1.

The Tables 1 and 2 show the results of our first experiment. In these tables, the average total execution time is shown from 1000 independent simulation trials. In addition, the 1.96 * standard deviation (estimated from the data) limits are shown in the brackets so that an approximate 95% confidence

interval may be constructed. In these experiments, the potential winner set is updated periodically with a batch of unprocessed bids. The winners are determined only once using the final potential winners set.

k (*batch size*)	Execution time (milliseconds)	
	without "initial acceptance test"	*with "initial acceptance test"*
1	152.0 (± 1.90)	26.9 (± 2.08)
5	76.8 (± 2.55)	9.1 (± 1.35)
10	65.6 (± 2.35)	6.1 (± 1.10)
20	62.2 (± 2.29)	7.2 (± 1.20)
50	67.1 (± 2.29)	7.5 (± 1.29)
200	82.6 (± 2.65)	8.8 (± 1.39)
1000	103.6 (± 2.94)	32.5 (± 1.57)
5000	120.6 (± 3.53)	120.9 (± 2.25)

Table 1: N=20, # Bids = 5000, maximum bid size = 20, # runs = 1000

k (*batch size*)	Execution time (milliseconds)	
	without "initial acceptance test"	*with "initial acceptance test"*
1	181.2 (± 4.04)	27.9 (± 3.12)
5	79.8 (± 2.59)	8.5 (± 1.53)
10	74.6 (± 2.69)	7.3 (± 1.31)
20	66.2 (± 2.43)	5.8 (± 1.06)
50	68.0 (± 2.27)	5.9 (± 1.08)
200	81.0 (± 2.43)	10.0 (± 1.43)
1000	103.0 (± 2.88)	30.3 (± 2.06)
5000	115.6 (± 3.12)	119.9 (± 2.86)

Table 2: N = 50, # Bids = 5000, maximum bid size = 20, # runs = 1000

As mentioned earlier, in the second experiment we compare the proposed algorithm using appropriate batch size and the initial acceptance test with the WCS 4.1 implementation. Recall that in the current implementation of winner determination in WCS 4.1 a process performs the winner computation every 90 seconds where it considers all the newly arrived bids (in the past 90 seconds) along with the ones that arrived earlier and performs the winner computation. This process continues throughout the duration of the auction. Thus a reasonable performance measure of interest is the total execution time, i.e., the sum of the execution times of each winner computation during the auction. To keep things simple, in our simulation we assume that exactly 20 bids arrive in each 90 second time interval, and we compare the performance of two algorithms under different scenarios arising by varying the total number of bids that arrive in the auction. The Tables 3 and 4 report the average total execution time (along with the 95% confidence interval) under different scenarios. Our algorithm collects a batch of k (value specified in each table) newly arriving bids and updates the set of potential winner bids. Both the algorithms compute the set of current winners after recieving 20 new bids.

Recall from Theorem 2 that if the bid size is uniformly distributed between 1 and N, then under the greedy auction rule, the average size of the potential winner set is $O(\log N)$. We verify this result in Table 5. We also run the experiment under knapsack rule and observe similar behaviour. Bid values

Total number of bids	Execution time in milliseconds	
	Proposed Algorithm	*WCS 4.1*
20	0.22 (± 0.022)	0.24 (± 0.071)
60	0.34 (± 0.025)	0.69 (± 0.122)
200	0.58 (± 0.037)	4.06 (± 0.284)
1000	1.49 (± 0.055)	110.0 (±19.6*)
5000	8.92 (± 0.078)	2440 (±102.8*)

Table 3: N = 20, k = 10, winner computation every 20 bids, maximum bid size = 20, # runs = 100,000. * In these cases only 500 readings were taken as computational time per reading was large.

Total number of bids	Execution time in milliseconds	
	Proposed Algorithm	*WCS 4.1*
20	0.25 (± 0.059)	0.26 (± 0.075)
60	0.49 (± 0.100)	0.54 (± 0.106)
200	0.77 (± 0.127)	3.87 (± 0.276)
1000	1.71 (± 0.176)	98.00 (± 14.70)*
5000	8.63 (± 1.176)	2476 (± 60.56)*

Table 4: N = 50, k = 20, winner computation every 20 bids, maximum bid size = 20, # runs = 100,000. * In these cases only 500 readings were taken as computational time per reading was large.

were assumed to be uniformly distributed between (2, 30) (the exact distribution is important under the knapsack rule). The experiments are conducted for different values of N. For each N considered, a total of $100 * N$ bids are generated and the potential winner set is determined. A total of 500 independent runs of this experiment are conducted and the average size and the 95% confidence interval of the number of bids in the potential winner set are reported in Table 5.

N	knapsack rules	greedy rules
1	1 (0)	1 (0)
5	3.8 (± 0.058)	3.8 (± 0.128)
20	7.68 (± 0.123)	7.6 (± 0.025)
100	13.236 (± 0.17)	12.64 (± 0.037)
200	15.91 (± 0.19)	14.85 (± 0.042)

Table 5: Average # of potential winner bids. Bid size $\sim$ uniform(1, N), bid value $\sim$ uniform(2, 30), # runs per N = 500.

7 Determining Winners using Distributed Sytems

When the number of bids is large, multiple co-located processors may be used to speed up the winner determination process. Also, if the bidders are geographically far apart, it may be useful to have a number of processors at different locations so that each bidder has a processor in close proximity. Our approach of identiying few potential winners from any given set of bids becomes useful in such cases. This approach is illustrated in a simple framework in this section.

Assume that there are $m+1$ processors. Set m of these processors to recieve the externally arriving bids. Thus externally arriving bids are partitioned into m streams (based on geography or any other criteria) and each stream is sent to a single processor. These processors maintain a list of potential winners from amongst the bids that have arrived and update this list with the new arrivals. Since they quickly filter out the loser bids these are referred to as 'filter' processors. Note that to determine current winners, the potential winner bids from all the filter processors need to be considered. This is achieved by sending the potential winner bids from each of the filter processor to the single 'master' processor that determines the current winners amongst all the potential winners and broadcasts it to the potential bidders. One issue of interest is the time between two consecutive deliveries of bids from any filter processor to the master processor. Let δ denote this time and for simplicity assume that it is same for each filter processor. The trade-off occurs as follows: Let $W_F(\delta)$ denote the waiting time of a bid at the filter processor. Let $p(\delta)$ denote the probability that it belongs to the list of potential winners at that processor. When that is the case, let $W_M(\delta)$ denote its waiting time at the master processor. Its expected overall waiting time equals:

$$E(W_F(\delta)) + p(\delta)E(W_M(\delta)).$$

Note that $E(W_F(\delta))$ is typically an increasing function of δ while $p(\delta)$ is a decreasing function of δ since larger the δ larger the number of bids that arrive and hence larger fraction of bids that are rejected. Also $E(W_M(\delta))$ is a decreasing function of δ as with increase in δ, the number of customers per unit time coming to the second queue decreases. An appropriate analytical framework needs to be developed to determine the optimal δ. However, that is beyond the scope of this work and we recommend it for future investigation.

8 Order Bids

At most of the internet sites conducting open-cry auctions, the bidders are given the option of placing a regular bid (i.e., bid where a unique bid value is specified) or placing an *order bid* (also called *proxy bid*), i.e., a software agent that bids on behalf of a human bidder. An order bid differs from the regular bid in the sense that it specifies the bidder's *maximum bidding limit* instead of a unique bid value. A naive implementation of the order bid functionality at a commerce webserver could result in a large number of bids being generated when two or more order bids engage in a bidding war. Not only does this waste the limited server resources, it also makes it difficult for the bidder to manage his bids. The following simple assumption on the auction rules makes winner determination straightforward, even when order bids are involved:

Assumption 5 *Under auction rules R, If $b \in W(R,S,N)$ for a given $val(b)$, then, keeping all else fixed, $b \in W(R,S,N)$ even if $val(b)$ is increased.*

This assumption is clearly satisfied by greedy and knapsack rules. It implies that to determine the winners in an auction that allows regular bids and order bids, each order bid may be treated as a regular bid with the maximum bidding limit treated as its bid value. Thus an order bid may only send a single bid. The issue of the amount that the order bid needs to pay is a complex one that, due to space considerations, is being addressed in a separate ongoing work ([1]). In ([1]), we also develop efficient winner determination algorithms under commonly used auction rules that combine the order bid and the regular bid processing.

9 Appendix

9.1 Proofs of Lemmas

Proof of Lemma 3: Consider **A2**. Let $m = \{j : X_j \geq Q(b_{new}) > X_{j+1}\}$. Then clearly bid $b_m \in G(greedy, Z, N)$ and $\hat{V}(Q(b_{new})) = val(b_m)$ and $\hat{Q}(Q(b_{new})) = Q(b_m)$. Now if $val(b_{new}) > val(b_m)$ or $val(b_{new}) = val(b_m)$ and $Q(b_{new}) > Q(b_m)$ then since b_{new} is above b_m in gsort and $X_m \geq Q(b_{new})$ it follows that $b_{new} \in G(greedy, Z + \{b_{new}\}, N)$. If these conditions are not satisfied then b_{new} is below b_m in gsort. Since $X_{m+1} < Q(b_{new})$ it follows that b_{new} is rejected as a loser bid. □.

Proof of Lemma 4: If $\bar{Q} \geq N/2$, $\alpha = 1$. Then, $\tau_1 = 1$ and the lemma holds. We therefore assume that $\bar{Q} < N/2$. Then, since for all i, $Q(b_i) \leq \bar{Q}$, we have $Y_1 = N - Q(b_1)$. Note that for $i \leq \tau_1$, $T_i = i$. Thus, for $2 \leq i \leq \tau_1$

$$\begin{aligned} Y_i &= Y_{i-1} - Q(b_i) \\ &= N - \sum_{j=1}^{i} Q(b_j). \end{aligned}$$

In particular, it follows that

$$EY_{\tau_1} = N - E(\sum_{j=1}^{\tau_1} Q(b_j)). \tag{17}$$

Using analysis similar to that used for proving Wald's identity (see, e.g., [4]) it is easily seen that

$$E(\sum_{j=1}^{\tau_1} Q(b_j)) = \sum_{j=1}^{\infty} E(Q(b_j))P(\tau_1 \geq j). \tag{18}$$

Due to Assumption 4, it is easily checked that $E(Q(b_j))$, for all j, is uniformly upper and lower bounded by a $\Theta(N^\alpha)$ term. Noting that $E(\tau_1) = \sum_{j=1}^{\infty} P(\tau_1 \geq j)$, it follows from (17) and (18) that

$$EY_{\tau_1} \leq N - E(\tau_1)\Theta(N^\alpha).$$

Now , since $Y_{\tau_1 - 1} > 2\bar{Q}$ it follows that $Y_{\tau_1} \geq \bar{Q}$ and hence $EY_{\tau_1} \geq \bar{Q}$. Therefore, we have

$$E(\tau_1) \leq (N - \bar{Q})/\Theta(N^\alpha).$$

and thus $E(\tau_1) = O(N^{1-\alpha})$. Similarly, we have

$$EY_{\tau_1} \geq N - E(\tau_1)\Theta(N^\alpha).$$

Note that $EY_{\tau_1} \leq 2\bar{Q}$. Thus,

$$E(\tau_1) \geq (N - 2\bar{Q})/\Theta(N^\alpha).$$

Therefore, for $N - 2\bar{Q} = \Theta(N)$, (15) follows. For the case when $\bar{Q} = N/2 - o(N)$ (a function $f(N)$ is said to be $o(N)$ if $\lim_{N\to\infty} f(N)/N = 0$), further proof is needed. Note that since $\tau_1 \geq 1$ and that $(Y_i : i \leq \tau_1)$ is a decreasing sequence, it follows that $EY_{\tau_1} \leq EY_1$. thus,

$$E(\tau_1) \geq (N - E(Y_1))/\Theta(N^\alpha).$$

Now $N - E(Y_1) = E(Q(b_1))$ and $E(Q(b_1)) = \Theta(N)$ since $\bar{Q} = N/2 - o(N)$. This completes the proof of (15). □.

The following lemma is useful in proving Lemma 5.

Lemma 6 *Under Assumption 4, there exists a constant $g < 1$ such that for all $\tau_1 \leq i \leq \tau_1 + \tau_2 - 1$:*

$$E(Y_{\tau_1+i}|Y_{\tau_1}, \tau_1) \leq g^i Y_{\tau_1}. \tag{19}$$

Proof: Let p denote the probability $P(\lceil \frac{3}{5} Y_{\tau_1+i} \rceil \leq Y_{\tau_1+i+1} \leq \lfloor \frac{4}{5} Y_{\tau_1+i} \rfloor | Y_{\tau_1+i}, \tau_1)$. We now lower bound this probability with a positive constant. First, recall that

$$Y_{\tau_1+i+1} = \max(Y_{\tau_1+i} - Q(b_{T_{\tau_1+i+1}}), Q(b_{T_{\tau_1+i+1}}) - 1).$$

Thus,

$$p \geq \sum_{x=\lceil 1/5 Y_{\tau_1+i} \rceil}^{\lfloor 2/5 Y_{\tau_1+i} \rfloor} P(Q(b_{T_{\tau_1+i+1}}) = x | Y_{\tau_1+i}, \tau_1).$$

Note that for $x \in (1, 2, \ldots, Y_{\tau_1+i})$, due to Assumption 4, $P(Q(b_{T_{\tau_1+i+1}}) = x | Y_{\tau_1+i}, \tau_1) \geq \beta/(\gamma Y_{\tau_1+i})$. Clearly, there exists a constant $c = \Theta(1)$ so that for any $w \geq c$ we have $\lfloor 2w/5 \rfloor - \lceil w/5 \rceil \geq w/6$. Then for $Y_{\tau_1+i} \geq c$, it follows that $p \geq \frac{1}{6}\beta/\gamma$. Now it is easily seen that for $Y_{\tau_1+i} \geq c$,

$$\begin{aligned} E(Y_{\tau_1+i+1}|Y_{\tau_1+i}, \tau_1) &\leq p\frac{4}{5} Y_{\tau_1+i} + (1-p) Y_{\tau_1+i} \\ &\leq Y_{\tau_1+i}(1 - \frac{1}{30}\beta/\gamma). \end{aligned}$$

For $Y_{\tau_1+i} < c$, it is clear that $Y_{\tau_1+i+1} \leq Y_{\tau_1+i} - 1 \leq (1 - 1/c) Y_{\tau_1+i}$. Thus, the result holds with $g = \max(1 - \frac{1}{30}\beta/\gamma, 1 - 1/c)$ by noting that

$$E(Y_{\tau_1+i+1}|Y_{\tau_1}, \tau_1) = E(E(Y_{\tau_1+i+1}|Y_{\tau_1+i}, \tau_1)|Y_{\tau_1}, \tau_1)$$

and proceeding recursively. □.

Proof of Lemma 5:

Note that $\tau_2 = \sum_{i=\tau_1+1}^{\infty} I(Y_i \geq 1)$ and thus

$$\begin{aligned} E(\tau_2|\tau_1, Y_{\tau_1}) &= \sum_{i=\tau_1+1}^{\infty} P(Y_i \geq 1|\tau_1, Y_{\tau_1}) \\ &\leq i^* + \sum_{i=\tau_1+1+i^*}^{\infty} E(Y_i|\tau_1, Y_{\tau_1}), \end{aligned}$$

where i^* is any positive integer and Markov inequality (see, e.g., [4]) is used to bound the probabilities by the corresponding expectations. Now from Lemma 6, we see that

$$E(\tau_2|\tau_1, Y_{\tau_1}) \leq i^* + Y_{\tau_1} \sum_{i=1+i^*}^{\infty} g^i.$$

Noting that $Y_{\tau_1} \leq 2\bar{Q}$ and setting $i^* = \log(2\bar{Q})/\log(1/g)$ (so that $i^* = \Theta(\log N)$ and $2\bar{Q}g^{i^*} = 1$), we get

$$E(\tau_2|\tau_1, Y_{\tau_1}) \leq \Theta(\log N) + \sum_{i=1}^{\infty} g^i,$$

and the result follows. □.

References

[1] Bassamboo A., Gupta M., Juneja S. and Kumar M., Efficient Order Bid Processing in Internet Open-Cry Auctions. Ongoing Work.

[2] Bertsimas D., Tsitsiklis J. N., *Introduction to Linear Optimization*, Athena Scientific Series in Optimization and Neural Computation, 1997.

[3] Cormen T. H., Leiserson C. E., and Rivest R. L., *Introduction to Algorithms*, The MIT Press, 1990.

[4] Feller W., *An Introduction to Probability Theory and its Applications*, Vol. II, Second Edition, John Wiley & Sons, New York. 1971.

[5] Horowitz E. and Sahni S., *Fundamentals of Computer Algorithms*, Computer Science Press, 1990.

[6] Iyengar A., Challenger J., Dias D., Dantzig P., Techniques For Designing High-Performance Web Sites, IBM Research Report, RC21324, Oct 1998.

[7] http://www.Internetnews.com.

[8] Leuker G., Average-Case Analysis of Off-line and On-line Knapsack Problems. *Mathematical Programming* 29 (2): 277-305, Nov 1998.

[9] Kumar M., Feldman S. I., *Internet Auctions*, http://www.ibm.com/iac/papers/auction_fp.pdf, 1998.

[10] Kumar M. and Feldman S. I., Business Negotiations on the Internet, *In Proc. Inet'98*, Geneva Switzerland, 1998.

[11] Marchetti-Spaccamela A. and Vercellis C., Stochastic On-line Knapsack Problems. *Mathematical Programming* 68(1): 73-104, 1995.

[12] Menasce D. A. and Almeida V. A. F., *Scaling for E-Business Technologies, Models, Performance and Capacity Planning*, Prentice Hall, 2000.