A Winner Determination Algorithm for Auction-Based Decentralized Scheduling

Chun Wang, Hamada H. Ghenniwa
Department of Electrical and Computer Engineering
The University of Western Ontario
Thompson Engineering Building Room 279, London, Ontario, Canada, N6A5B9
+1 519 6612111-80334, +1 519 6612111-88262
cwang28@uwo.ca, hghenniwa@eng.uwo.ca

Weiming Shen
Integrated Manufacturing Technologies Institute,
National Research Council Canada
800 Collip Circle, London, Ontario, Canada
N6G4X8
Tel: +1 519 4307134
Weiming.shen@nrc.ca

ABSTRACT
This paper presents a formulation and an algorithm for the winner determination problem in auction-based decentralized scheduling. Without imposing a time line discretization, the proposed approach allows bidders to bid for the processing of a set of tasks under release time and due date constraints using an expressive bidding language designed for decentralized scheduling. The proposed winner determination algorithm uses a depth first branch and bound search. The search branches on bids and a constraint directed scheduling procedure is used at each node to verify the feasibility of the allocation. Experiments against a commercial optimization package, CPLEX 10.0, show that the proposed algorithm is more than an order of magnitude faster on average over a set of winner determination problems of decentralized scheduling generated based on a suite of job shop constraint satisfaction benchmark problems previously developed in the literature.

Categories and Subject Descriptors
I.2 [**Artificial Intelligence**]: General

General Terms
Economics, Algorithms

Keywords
Winner determination, combinatorial auctions, decentralized scheduling, constraint directed search

1. INTRODUCTION
Decentralized scheduling problems are characterized with distributed information about the overall problem and multiple (conflicting in many cases) objectives of agents. In a decentralized scheduling problem, autonomous agents, representing individuals, enterprises, or computational devices, have tasks that need to be completed during a specific time period and they compete with each other for the processing times of the resources. An agent in a decentralized scheduling problem usually has constraints over its set of tasks, such as release time, deadline, and precedence constraints. In addition, a task may only be processed on a specific subset of resources. Therefore, an agent is only interested in the combinations of specific time periods of resources, which satisfy the constraints of its tasks. That is, for the agents, there are complementarities between different processing time periods of resources.

If the processing times available on resources are considered as goods to be sold, a decentralized scheduling problem can be mapped to a combinatorial allocation problem which can be solved using combinatorial auctions. An obvious issue with the mapping is how to map the processing times of resources, which is divisible, to indivisible distinct items required in combinatorial allocation problems. One possible approach is to impose a discretization on the time windows of resources to be scheduled and treat the time slots generated by the discretization as distinct items. However, this discretization approach can generate a large number of items if the time windows in question are big. For example, a one week time window on 10 resources can be discretized into more than 1 million time slots if the time accuracy we need is in minutes (which is a practical requirement in many application domains). Generally speaking, in combinatorial auction the number of possible bids is exponential in the number of the items to be sold. A large number of items can inflict heavy burdens on both agents (in terms of bids evaluation) and the auctioneer (in terms of winner determination).

This paper presents a formulation and an algorithm for the winner determination problem (WDP) in decentralized scheduling. As one cannot hope for a general-purpose algorithm that can efficiently solve every instance of the WDP [6], we develop a domain specific winner determination algorithm using branch & bound and constraint-based node feasibility validation to overcome the restriction of having to discretize resource processing times into units. The rest of the paper is organized as follows. Section 2 compares the proposed scheduling based WDP formulation with a general WDP formulation in the context of decentralized scheduling. In Sections 3 and 4, we propose a winner determination algorithm for the scheduling based WDP formulation and present comparative results on a suite of test problems. The algorithm is further refined in Section 5. Section 6 presents conclusion and future research directions.

AAMAS'07, May 14-18, 2007, Honolulu, Hawai'i, USA.

978-81-904262-7-5 (RPS) ©2007 IFAAMAS

2. WDP FORMULATIONS FOR DECENTRALIZED SCHEDULING

2.1 A General WDP Formulation

To formulate the WDP as an integer program, let N be the set of bidders and M the set of distinct objects. For every subset B of M let $v_j(B)$ be the price that agent $j \in N$ has announced she is willing to pay for B. Let $x_j(B) = 1$ if the bundle $B \subseteq M$ is allocated to $j \in N$ and zero otherwise.

$$\max \sum_{B \subseteq M} \sum_{j \in N} x_j(B) v_j(B)$$

$$\text{s.t.} \quad \sum_{B \subseteq M} x_j(B) \le 1, \qquad \forall j \in N$$

$$\sum_{B \subseteq M, i \in B} \sum_{j \in N} x_j(B) \le 1, \quad \forall i \in M$$

$$x_j(B) = 0,1, \qquad \forall j \in N, B \subseteq M$$

This formulation is called *combinatorial auction problem* (CAP) in [17]. To apply CAP to decentralized scheduling problems, one can impose a discretization of time line into finite slots and treat the time slots as a set of distinct items to be allocated to agents. Figure 1 illustrates how this idea is implemented through a simple decentralized scheduling problem, which is a slight modification form the Factory Scheduling Economy example adopted in [18]. In Figure 1, an unscheduled day shift of a factory is divided into eight one-hour time slots, labeled 9:00 to 16:00 according to their respective end times. Slots are treated as distinct items that can be allocated for the production of customer orders. Assume each customer agent has one single-operation job to be completed. An agent's job is defined by its duration (length), its release time (the time when the job is available for processing), its deadline, and the price (expressed in dollars) the agent places on the job. To complete its job, the agent must acquire a number of slots no less than the length (not necessarily contiguous), within its feasible time window (the time period between its release time and its deadline).

For this example, we may construct an auction in which an agent can bid any combination of timeslots (with the sum of slots equal or greater than the length of its job) within its feasible time window. For example, Agent 1 can submit 4 XOR bids which are $\{9,10,11\}$, $\{10,11,12\}$, $\{9,11,12\}$ and $\{9,10,12\}$. In this case, the WDP is to allocate time slots to agents such that no slot is allocated to more than one agent, no agent receives more than one bid and the revenue (sum of winning agents' prices) is maximized.

In the above example, the number of feasible bids that an agent can submit is restricted by the release time, the deadline and the number of time slots that the agent needs to process its job. However, we have assumed that job pre-emption is allowed. We may further restrict the feasible bids by removing this assumption, which means that an agent can bid only on time slots that are adjacent. As shown in Theorem 1, the problem remains NP-hard even with above and other restrictions.

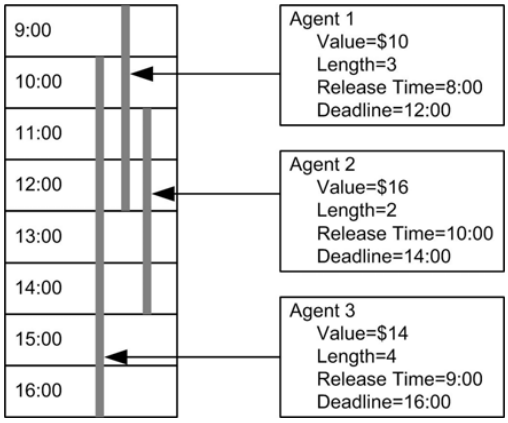

Figure 1. An Example of Decentralized Scheduling Problem

Theorem 1 *The CAP remains NP-hard even if we restrict to instances where the items to be sold are a set of linearly ordered time slots of a single resource; a bid has only one operation to be scheduled on the resource between its release time, r_j and deadline, d_j; pre-emption is not allowed; all operations have identical processing time τ; and $d_j - r_j - \tau \le 2$.*

Proof. To show that the CAP with restrictions is NP-hard, we prove its decision version is NP-complete by constructing a polynomially computable reduction from the following variation of the job interval selection problem (JISP) which is NP-complete [7]. Given a set of $J = \{J_1, J_2, ..., J_n\}$ jobs in a JISP, each of which requires the same execution time τ, and J_j may be started for execution at any of a set of given discrete starting times $S_j = \{s_j, ..., s_j + k_j\}$ where $k_j \le 2$, $1 \le j \le n$, we can construct an instance of the CAP as follows. Bid j has job J_j to be executed with the release time $r_j = s_j$ and deadline $d_j = s_j + k_j + \tau$. v_j is the price that agent j is willing to pay for any bundles that satisfy its constraints. Because $k_j \le 2$, $d_j - r_j - \tau \le 2$ is satisfied. Let $V = \sum_{j=1}^{n} v_j$. To answer the question if the CAP has a solution with revenue V is equivalent to solving the NP-complete IJSP ■.

2.2 Time Window Scaling Problem

The time discretization approach to WDP in decentralized scheduling is quite straightforward. However, it can lead to increased number of bids to be submitted when the feasible time window of an agent increases. To see why, suppose that an agent j has a job with n_j operations to be processed in a feasible time window W with release time r_j and deadline d_j, $W = d_j - r_j$.

For each operation o_k $(k = 1,...,n_j)$ a processing time p_k $(k = 1,...,n_j)$ is given. There are linear precedence constraints among operations, that is o_k must precede o_{k+1}. To schedule the set of operations in W, 3 constraints have to be satisfied:

$$S_{k-1} + p_{k-1} \le S_k \qquad for\ 1 < k \le n_j \tag{1}$$

$$S_1 \ge r_j \tag{2}$$

$$S_{n_j} \le d_j - p_{n_j} \tag{3}$$

where S_k is the starting time of o_k.

Given the constraints, the number of feasible schedules in time window W is calculated by the following formula:

$$\sum_{S_1=0}^{W-p_1-p_2...p_{n_j}} \sum_{S_2=p_1+S_1}^{W-p_2-p_3...p_{n_j}} .... \sum_{S_{n_j}=p_{n_{j-1}}+S_{n_{j-1}}}^{W-p_{n_j}} 1 \tag{4}$$

According to (4), the number of feasible schedules in time window is not exponential in W because an upper bound, $(W - n_j + 1)^{n_j}$, can be obtained by relaxing constraint (1) and set $p_1 = p_2 = ...p_{n_j} = 1$. However, the feasible bids that an agent can submit still increase drastically when W increases. According to Theorem 1, the CAP remains NP-hard with scheduling domain restrictions. Increased number of bids leads to larger problem size which demands exponentially growing computation time. We refer this problem as *Time Window Scaling* problem in the CAP formulation for decentralized scheduling.

The *Time Window Scaling* problem is caused by the time discretization when mapping processing times of resources to distinct items in CAP. However, in classical scheduling models, the time line to be scheduled is not discretized. In these models, the extension of a job's feasible time window only changes the values of constraint parameters. The problem size will not be significantly impacted[1]. Based on this observation, we propose a new WDP formulation for auctions designed for the decentralized scheduling problem, which does not impose a discretization on the time line to be scheduled. We call it the CAP for scheduling (SCAP).

2.3 SCAP

In SCAP, instead of imposing a finite time discretization on the system, we provide an expressive bidding language that allows bidders to bid for the processing of a set of tasks under release time and deadline constraints. In this language a bid is a 3-tuple $Bid = \langle operations, constraints, price \rangle$. Constraints may have several entries. We only consider release time, deadline and precedence constraints in this paper. We demonstrate the construction of a SCAP in the context of a specific class of decentralized scheduling problems described as follows.

[1] The problem size will slightly go up because binary encoding is used to calculate the size of a problem.

Consider a decentralized scheduling problem with a set of n bids. Each bid j $(j = 1,...,n)$ requires the processing of a sequence of operations $o_{j,k}$ $(k = 1,...,n_j)$. An operation $o_{j,k}$ $(1 \le j \le n, 1 \le k \le n_j)$ has a specified processing time, $p_{j,k} \in R^+$, and its execution requires the exclusive use of a designated resource for the duration of its processing. If $o_{j,k}$ and $o_{\hat{j},\hat{k}}$ need to be processed on the same resource $q_{j,k,\hat{j},\hat{k}} = 1$, otherwise $q_{j,k,\hat{j},\hat{k}} = 0$. All the operations of bid j can only begin after its release time r_j and must be finished before its deadline d_j. There are precedence constraints among operations of a bid. The WDP involves the selection of a subset of bids such that, the sum of bid prices are maximized and all constraints are satisfied. Using the following variables:

$S_{j,k}$ the starting time of the operation k of bid j.

$$Z_j = \begin{cases} 1 & \text{if b id } j \text{ wins} \\ 0 & \text{otherwise.} \end{cases}$$

$$Y_{j,k,\hat{j},\hat{k}} = \begin{cases} 1 & \text{if } o_{j,k} \text{ is performed before } o_{\hat{j},\hat{k}} \\ 0 & \text{otherwise, } j \ne \hat{j} \end{cases}$$

the WDP can be formulated as follows.

$\max \sum_{j \in T} Z_j v_j$, where v_j is the price of bid j

Subject to

$$S_{j,1} \ge r_j Z_j \qquad j = 1,...,n \tag{5}$$

$$S_{j,n_j} + p_{j,n_j} \le d_j + H(1 - Z_j) \qquad j = 1,...,n \tag{6}$$

where H is a large finite positive number.

$$S_{j,k-1} + p_{j,k-1} - S_{j,k} \le H(1 - Z_j) \qquad j = 1,...,n \quad 1 < k \le n_j \tag{7}$$

$$S_{j,k} + p_{j,k} - S_{\hat{j},\hat{k}} + Hq_{j,k,\hat{j},\hat{k}} + HZ_j + HZ_{\hat{j}} + HY_{j,k,\hat{j},\hat{k}} \le 4H, \quad j = 1,..n,\ \hat{j} = 1,...,n,\ j \ne \hat{j},\ 1 \le k \le n_j, 1 \le \hat{k} \le n_{\hat{j}} \tag{8}$$

$$Y_{j,k,\hat{j},\hat{k}} + Y_{\hat{j},\hat{k},j,k} + 2H \ge 1 + HZ_j + HZ_{\hat{j}}, \quad j = 1,..n,\ \hat{j} = 1,...,n,\ j \ne \hat{j},\ 1 \le k \le n_j, 1 \le \hat{k} \le n_{\hat{j}} \tag{9}$$

$$Y_{j,k,\hat{j},\hat{k}} + Y_{\hat{j},\hat{k},j,k} + HZ_j + HZ_{\hat{j}} \le 1 + 2H, \quad j = 1,..n,\ \hat{j} = 1,...,n,\ j \ne \hat{j},\ 1 \le k \le n_j, 1 \le \hat{k} \le n_{\hat{j}} \tag{10}$$

$$Y_{j,k,\hat{j},\hat{k}} \in \{0,1\},$$
$$j = 1,...n,\ \hat{j} = 1,...,n,\ j \neq \hat{j}, 1 \le k \le n_j, 1 \le \hat{k} \le n_{\hat{j}} \quad (11)$$

$$Z_j \in \{0,1\}, \qquad j = 1,...n \quad (12)$$

$$S_{j,k} \ge 0, \qquad j = 1,...n, 1 \le k \le n_j \quad (13)$$

The set of constraints (5) and (6) ensure that the operations of a bid do not start before the release time and finish after the deadline. The set of constraints (7) ensure that an operation does not start before the previous operation of the same bid is completed. The set of constraints (8), (9) and (10) ensure that at most one operation can be processed by a particular resource at a time. Constraints (11), (12), (13) are non-negative and integer constraints. Clearly that, in formulation SCAP, the extension of a bid's feasible time window will not affect the number of the bids submitted.

3. WINNER DETERMINATION ALGORITHM

While the SCAP formulation does not suffer from the *Time Window Scaling* problem, however, it introduces a challenging issue: the feasibility validation of solutions. For the CAP formulation, as long as any two winning bids do not share an item, the solution is feasible. However, in SCAP, validating the feasibility of a solution is equal to answering the question: given a set of bids with constraints, does a schedule exist that allocates the operations of bids on the resources, such that all constraints are satisfied? This decision problem is actually a job shop Constraint Satisfaction Problem (CSP) [16]. This problem is known to be NP-complete [5]. Among proposed approaches to the problem, constraint directed search provides good results.

3.1 Constraint-Based Feasibility Validation

We describe the feasibility validation algorithm based on a disjunctive graph representation of the job shop CSP. Consider a directed graph G with a set of nodes N and two sets of arcs A and B. A node corresponds to an operation $o_{j,k}$ and its designated resources $u(o_{j,k})$, denoted by $(u(o_{j,k}), o_{j,k})$. $u(o_{j,k})$ is a set of resources which has the capability of processing $o_{j,k}$. In job shop CSP, $|u(o_{j,k})| = 1$. The so-called conjunctive (solid) arcs A represent the precedence constraints of two operations belong to the same job. If $arc(u(o_{j,k}), o_{j,k}) \rightarrow (u(o_{j,\hat{k}}), o_{j,\hat{k}})$ is part of A, then $o_{j,k}$ has to be processed on $u(o_{j,k})$ before $o_{j,\hat{k}}$ is processed on $u(o_{j,\hat{k}})$. Two operations that belong to two different jobs and that have to be processed on the same resource are connected to one another by two so-called disjunctive (broken) arcs that go in opposite directions. The disjunctive arcs B form m cliques of double arcs, one clique for each resource. All operations in the same clique have to be done on the same resource. All arcs emanating from a node have as length the processing time of the operation that is represented by that node. In addition, there is a source

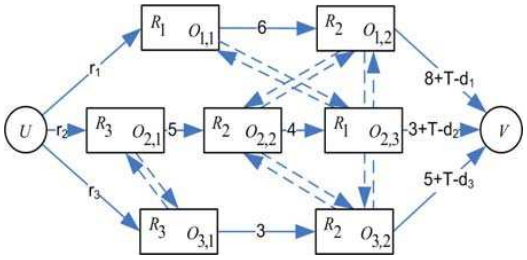

Figure 2. Disjunctive graph for a job shop CSP

U (representing time zero) and a sink V (representing an upper bound T of the minimal makespan of the problem), which are dummy nodes. The first operation of each job is connected to U by a conjunctive arc emanating from U with as length the release time of that job. The last operation of each job is connected to V by a conjunctive arc emanating to V with as length of $p_{j,n_j} + T - d_j$. This graph is denoted by $G = (N, A, B)$. A feasible schedule of job shop CSP corresponds to a selection of one disjunctive arc from each pair such that the resulting directed graph $\overline{G} = (N, A, \overline{B})$ is acyclic and the longest path in $\overline{G}$ from the source U to the sink V is not longer than T, where $\overline{B}$ denotes the subset of the selected disjunctive arcs. Figure 2 shows an example of the disjunctive graph representation of a small job shop CSP.

If we model each disjunctive arc pair in B as a variable and the two opposite directions as two possible values, this constraint satisfaction problem can be solved by a constraint directed backtrack search procedure in which the solution is incrementally extended through the repeated binding of a direction to unconstrained disjunctive arc pairs. Usually, a constraint directed search procedure consists of propagators, heuristic-commitment techniques and retraction techniques. The feasibility validation algorithm integrates Constraint-Based Analysis (a propagator, developed in [3]), Precedence Constraint Posting (a commitment heuristic, developed in [16]), and a chronological backtracking. The validation function is invoked by calling CHECK-FEASIBILITY (*bids*).

Algorithm1

function CHECK-FEASIBILITY (*bids*) **returns** *pass* or *failure*

$N \leftarrow \{operations\ in\ bids\}$

$A \leftarrow \{conjunctive\ arcs\ in\ bids\}$

$B \leftarrow \{disjunctive\ arcs\ in\ bids\}$

$G = (N, A, B)$

return BACKTRACKING (G)

function BACKTRACKING (G,) **returns** *pass* or *failure*

if inconsistent assignments detected in G

then return failure.

if $\{uncommitted\ \ pairs\ in\ B\} = \phi$ **then return** *pass*

/*Constraint propagating*/

CONSTRAIN-BASED-ANALYSIS (G)

if $\{uncommitted\ \ pairs\ in\ B\} = \phi$ **then return** *pass*

/*Variable ordering*/

$pair \leftarrow PCP(\{uncommitted\ \ pairs\ in\ B\})$

/*Value ordering*/

direction-sequence =*PCP* (*pair*)

for each *direction* **in** *direction-sequence* **do**

commit *direction* to *pair*

$result \leftarrow$ BACKTRACKING (*G*)

if $result \neq failure$ **then return** *pass*

de-commit *direction* to *pair*

return *failure*

3.2 The Branch and Bound Algorithm for Scheduling (BBS)

3.2.1 Branching structure

A branch bound based algorithm for a WDP can branch on items [14][4] or branch on bids [15]. Since there are not items in the SCAP formulation, BBS branches on bids. The algorithm is a depth-first tree search. Figure 3 depicts a 3 bids search tree of BBS. The search starts with an empty temporal schedule, called $TEMP$. Along the path $TEMP$ is expended by adding more bids form AV, which is a set that constraints available (not winning) bids. The best $TEMP$ found so far is $TEMP^*$. sum is the revenue of $TEMP$, which is the summation of prices of bids in $TEMP$, and sum^* be the revenue of $TEMP^*$. h is an upper bound on how much the bids in AV can contribute. The search is invoked by calling BRANCH-BOUND-SCHEDULING (*bids*).

Algorithm 2

function BRANCH-BOUND-SCHEDULING (*bids*) **returns** *solution*

$TEMP = \phi, TEMP^* = \phi$, and $sum^* = 0$

RECURSIVE-BRANCHING (*bids* , 0)

return $TEMP^*$

function RECURSIVE-BRANCHING (AV , sum)

if $sum > sum^*$ **then** $sum^* \leftarrow sum$, $TEMP^* \leftarrow TEMP$

if $AV = \phi$ **then return**

$h \leftarrow \sum_{bid \in AV} priceOf(bid)$

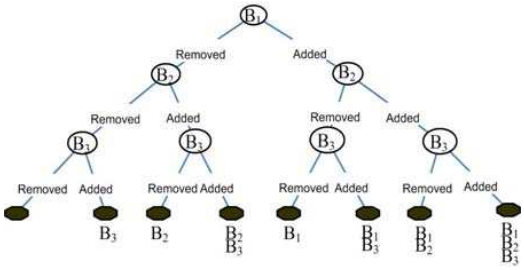

Figure 3. A BBS search tree of 3 bids

If $sum + h \le sum^*$, **then return**

$bid \leftarrow$ SELECT-UNASSIGNED-BID (AV)

$TEMP = TEMP \bigcup \{bid\}$, $AV = AV - \{bid\}$

$sum \leftarrow sum + priceOf(bid)$

if CHECK-FEASIBILITY ($TEMP$) returns pass

then RECURSIVE-BRANCHING (AV , sum)

$TEMP \leftarrow TEMP - \{bid\}, sum = sum - priceOf(bid)$

RECURSIVE-BRANCHING (AV , sum)

add *bid* to AV , **return**

3.2.2 Bid ordering heuristics

Search speed of BBS can be improved by incorporating bid ordering heuristics in the function SELECT-UNASSIGNED-BID (AV). We give bids that contribute a lot to the revenue, and leave more possibilities to the potential contribution of other bids more weight. Clearly, the price of bids is a measure of contribution to the revenue. We use the *flexibility* of a bid to measure the possibility that it leaves to other bids to contribute. For a bid j, the *flexibility* of j is defined as $flex_j = slack_j / (d_j - r_j)$ where $slack_j = d_j - r_j - \sum_{k=1}^{n_j} p_j$. Clearly, $slack_j$ represents the free space in bid j's time window ($d_j - r_j$), in which bid j's operations can be feasibly shuffled around. A bid with high *flexibility* can easily adjust the processing of its operations to accommodate other bids in the schedule. At a search node, we select a bid that maximizes $v_j \exp(flex_j / Q)$ to be added to $TEMP$. Q is the scaling parameter that can be determined empirically. Intuitively, a large Q gives too much preference to bids with high prices. On the other hand, a smaller Q gives more preference to bids with high *flexibilities.*

4. A COMPUTATIONAL STUDY

4.1 Design of the Test Data

Since common combinatorial auction benchmarks distributions, such as those presented in [14][10], are developed for the CAP

Table 1. Configuration of the test problem set

Group	BK	RG	P	Bids	Instances
1	1	0.4	0	5-10	60
2	1	0.5	0	5-10	60
3	2	0.5	0	5-10	60
4	2	0.4	0	5-10	60
5	1	0.5	1	5-10	60
6	1	0.4	1	5-10	60
7	1	0.3	1	5-10	60

formulation, we design our decentralized scheduling test problems based on a suite of job shop CSP benchmark problems developed in [13] . Two parameters were adjusted to cover different scheduling conditions in [13]. The first one is a range parameter, *RG*, which controls the distribution of job deadlines and release times. The second is a bottleneck parameter, *BK*, which controls the number of major bottleneck resources. In our decentralized scheduling test problems, we have introduced a third parameter, *P*, to control the distribution of the prices of bids. Deadlines are randomly drawn from a uniform distribution $MU(1-RG,1)$, where $U(a,b)$ represents a uniform probability distribution between a and b , M is an estimate of the minimum makespan of the problem, which is determined by the average duration of all operations and the average duration of the operations requiring bottleneck resources. This estimate was first suggested in [11]. Similarly, release times are randomly drawn from a uniform distribution of the form: $MU(0,RG)$. The price of bid j is randomly drown from a uniform distribution on $U(Pdu_j, du+Pdu_j)$, where du is the average duration of all bids, and du_j is the duration of bid j . In the original job shop CSP benchmark design, M is inflated to $(1+S)M$, where S is called slack parameter which is a function of *RG* and *BK*. The rational of using S is to make sure that most problems generated remain feasible (including all jobs). In our test problem design, we set $S=0$ because if all bids can be included in an optimal solution, the winner determination problem becomes trivial for the BBS. By considering different values of the parameters, seven groups of 60 problems of different sizes (number of bids) were randomly generated. The details are summarized in TABLE 1.

4.2 Experimental Results

This section reports the performance of BBS over the decentralized scheduling test problem set against CPLEX10.0. It was reported in [1] that as a general-purpose integer programming package, CPLEX 6.5 performs very well for many of the common benchmarks distributions (comparable to the special-purpose winner determination algorithms, such as those in [4][14]). A detailed comparison between CPLEX8.0 and a recently developed sophisticated algorithm CABOB can be found in [15].

We tested BBS and CPLEX 10.0 on the first six groups of the test problems. There are 60 problem instances in a group. These instances are divided into 6 sub-groups according to the number of bids an instance has. The number of bids can be seen as a measure of the size of an instance. Each sub-group has 10 instances of the same size. The instance size scales from 5 to 10 in a group. We have imposed a 1000 second time limit for both algorithms. All the instances under size 8 can be solved by both algorithms within the time limit. When the size scales to 9 and 10, some of them cannot be solved within the time limit. For a sub-group, if the number of the instances solved within the time limit is less than 8, we do not consider the sub-group in our comparison.

Figure 4 shows the running times of BBS and CPLEX10.0 over the six groups of the test problems. On average BBS is around an

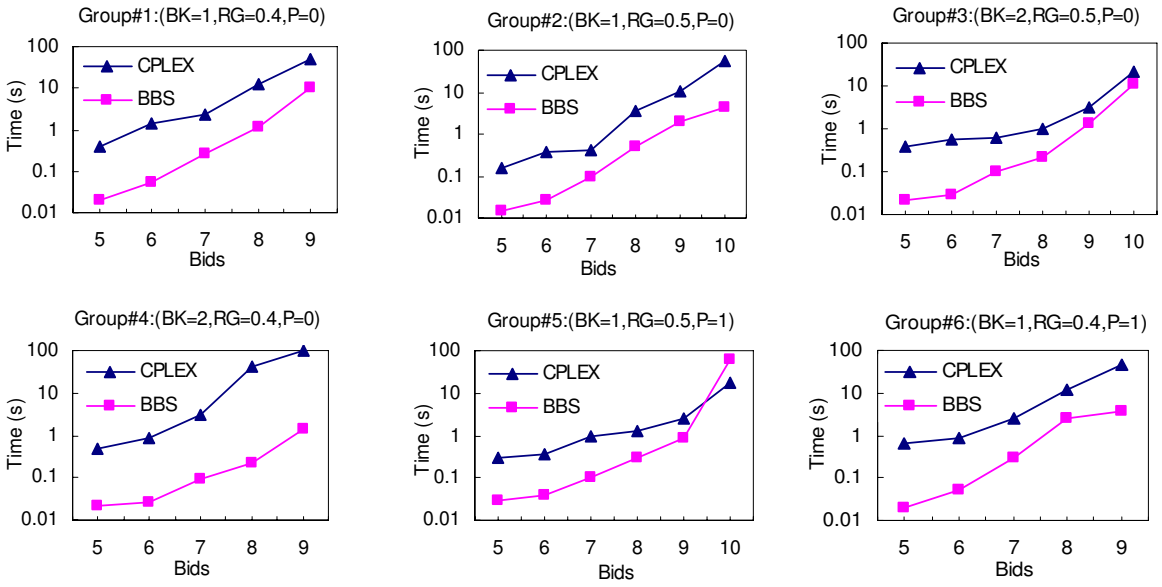

Figure 4. Run times on 6 groups of test problems

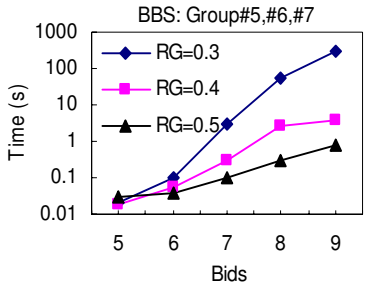

Figure 5. BBS run times on 3 group of test problem with different RG values

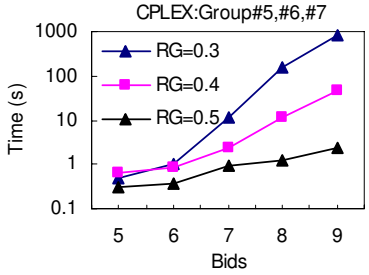

Figure 6. CPLEX Run times on 3 groups of test problems with different RG values

order of magnitude faster than CPLEX10.0. However, there are exceptions. For the instances of size 9 and 10 in groups #3 and #5, CPLEX10.0 is close and some times faster than BBS. On the other hand, for the instances of all sizes in group #4, BBS is more than one order (around two orders in some cases) of magnitude faster than CPLEX10.0. Figure 5 and Figure 6 depict the performance of BBS and CPLEX10.0 over three problem groups. Instances in these groups were generated with different values of *RG*. Values for other parameters were kept same among groups. It is observed that BBS and CPLEX10.0 have similar trends in Figure 5 and 6. That is, instances with tight constraints (bigger *RG*) tend to be solved quicker.

5. CONSTRAINT-BASED PRUNING SCHEME

While BBS performed quite well over the tested problem set, it does not in fact utilize all of the information provided by the SCAP formulation. As shown in Figure 7, the solutions of a SCAP with n ($n = 4$ in Figure 7) bids, can be organized using an n-level structure. A solution at level k $(1 \le k \le n)$, contains exactly k bids and the number of solutions at level k is $n!/(n-k)!\,k!$. BBS adopts a top down approach (same as CABOB in [15]). In order to find a good solution quickly, it starts with bigger size solutions (from right to left in the search tree shown in Figure 3) trying to include as many bids as possible in the beginning. If a solution that includes all bids happens to be feasible, the search finds the optimal solution without any backtracks. However, if a solution (includes all or a high percentage of bids) is not feasible, in order to backtrack to alternative solutions, BBS has to first prove that infeasibility. That is, it needs to prove the infeasibility of an underlying job shop CSP (which is NP-hard) with a bigger size. To prove that a CSP is infeasible usually takes longer because each possible solution has to be explicitly or implicitly enumerated. This may lead to bad *any time performance* for BBS. We have observed in our experiments that BBS goes slow in the early stages of search.

In addition, BBS cannot effectively utilize the results of previously solved job shop CSPs to prune search space. For example, suppose that B_2 and B_3 need to use a bottle neck resource at the same time and this conflict cannot be resolved according to their constraints. In other words, B_2 and B_3 cannot coexist in a feasible solution. It becomes obvious that any solutions that include B_2B_3 is not feasible as well (such as $B_1B_2B_3$ and $B_2B_3B_4$). Because BBS starts with bigger size solutions, it may encounter $B_1B_2B_3$ and/or $B_2B_3B_4$ before B_2B_3. BBS would not know what caused the problem until B_2B_3 is solved. In light of this observation, we have developed an improved constraint-based pruning scheme to further prune the search space using a simple constraint propagator: any solution that includes an infeasible solution is infeasible. The algorithm starts with smaller size solutions and tries to identify conflicts at early stage and use the conflicts information to prune the search space. However this pruning scheme does not always leads to shorter computation time as for many problems there are no hard conflicts between bids. Even if there are some, they may have been pruned by the upper

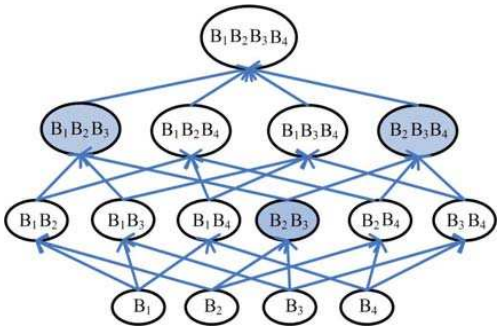

Figure 7. Improved pruning in BBS

bounding scheme of BBS before they are identified. Preliminary experiments show that for some moderate size problems that usually take BBS 40-80 seconds, if a certain percentage (above 40%) of job shop CSPs is pruned by the scheme during search, usually similar percentage of computation time can be saved. If the percentage is too low the constraint-based pruning scheme can cause longer computation time because there is a cost associated with the propagating of infeasibility constraints. We leave further investigation of this pruning scheme to future research.

6. RELATED WORKS

Limited attention has been devoted to auctions for decentralized scheduling. Wellman et al. [18] investigated auction protocols for decentralized scheduling. They imposed a discretization of time into finite slots, while our approach allows agents to use an

expressive bidding language to bid for the processing of a set of tasks under certain constraints. In [12], Parkes and Ungar presented an auction-based method for decentralized train scheduling. The bidding language designed in the train scheduling auction avoids use of discrete time slots. Bids are expressed by specifying a price to enter a track line and a time window. The winner determination problem was formulated with mixed integer programming, with many domain-specific constraints, and solved with CPLEX. No domain-specific winner determination algorithm was proposed. In [2] auctions over tasks with complex time constraints and interdependencies were proposed. The problem was not to schedule resources the agent has, but to produce a schedule of tasks that other agents would do. The objective was to optimize the expected customer's utility before bids are submitted and schedules are finalized. In [8], combinatorial auctions were applied to the job shop scheduling problem. The focus of this work was to investigate the links between combinatorial auctions and Lagrangean relaxation, and to design auctions based on the Lagrangean based decomposition. A "schedule selection game" was presented in [9] for collaborative production scheduling. The emphasis of this work was on the incentive compatibility of the mechanism rather than winner determination. Recently proposed winner determination algorithms can be found in [4][14][15]. These algorithms target the CAP and cannot be directly applied to our winner determination problem formulation for the decentralized scheduling problems.

7. CONCLUSION

The objective of this research is to investigate how the domain specific properties of scheduling problems can facilitate efficient winner determination algorithms. We have presented a formulation and an algorithm for the winner determination problem of auction-based decentralized scheduling. The proposed method takes advantage of the fact that dedicated scheduling techniques are more efficient than general mixed integer programming methods. We use a constraint-directed scheduling algorithm to verify the feasibility of the allocation at each node of the branch and bound search. We have also proposed a constraint-based pruning scheme which uses the domain-specific heuristics to further prune the search space. The current formulation restricts bids to one dimension, which is price only. In our future work, we plan to extend this work to multi-attribute winner determination problems in decentralized scheduling.

8. REFERENCES

[1] Andersson, A., Tenhunen, M., and Ygge, F. Integer programming for combinatorial auction winner determination. In *Proceedings of the Fourth Internet. Conf. Multi-Agent Systems (ICMAS)*, IEEE Computer Society, Boston, MA, 2000, 39–46.

[2] Babanov, A., Collins, J., and Gini, M. Scheduling tasks with precedence constraints to solicit desirable bid combinations. In *Proc. of the Second Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, Melbourne, Australia, July 2003, 345-352.

[3] Erschler, J., Roubellat, F., and Vernhes, J.P. Characterizing the set of feasible sequences for n jobs to be carried out on a single machine. *European Journal of Operational Research*, 4, 1980, 189-194.

[4] Fujishima, Y., Leyton-Brown, K., and Shoham, Y. Taming the Computational Complexity of Combinatorial Auctions: Optimal and Approximate Approaches. In *Proceedings of IJCAI-99*, Stockholm, 1999.

[5] Garey, M.R., and Johnson, D. S. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W. H. Freeman Company, 1979.

[6] Lehmann, D., Muller, R., Sandholm, T., The Winner Determination Problem. In *Combinatorial Auctions*, edited by Peter Cramton, Yoav Shoham, and Richard Steinberg, 2006.

[7] Keil, J. M., On the complexity of scheduling tasks with discrete starting times. *Operations Research Letters*, 12, 1992, 293-295.

[8] Kutanoglu, E., Wu, S. D. On combinatorial auction and Lagrangean relaxation for distributed resource scheduling. *IIE Trans.,* 31, 9 (Sept. 1999), 813-826.

[9] Kutanoglu, E., Wu, S. D. Incentive compatible, collaborative production scheduling with simple communication among distributed agents. *International Journal of Production Research*, 44, 3, 2006, 421-446.

[10] Leyton-Brown, K., Pearson, M., Shoham, Y. Towards a universal test suite for combinatorial auction algorithms. In *Proc. ACM Conf. Electronic Commerce (ACM-EC)*, Minneapolis, MN. ACM, New York, 2000, 66–76.

[11] Ow, P.S. Focused scheduling in proportionate flowshops. *Management Science*, 31, 1985, 852-869.

[12] Parkes, D. C. and Ungar, L. An Auction-Based Method for Decentralized Train Scheduling. In *Proceedings of 5th International Conference on Autonomous Agents (AGENTS-01)*, Montreal, Quebec, Canada, 2001, 43-50.

[13] Sadeh, N., and Fox, M., Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem. *Artificial Intelligence*, 86, 1996, 1-41.

[14] Sandholm, T. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135, 2002, 1-54.

[15] Sandholm, T., Suri, S., Gilpin, A. and Levine, D. CABOB: A Fast Optimal Algorithm for Winner Determination in Combinatorial Auctions. *Management Science*, 51, 3, 2005, 374-390.

[16] Smith, S.F., and Cheng, C. Slack-Based Heuristics for Constraint Satisfaction Scheduling. In *Proceedings of 11th National Conference on Artificial Intelligence.* Washington D.C., July 1993.

[17] de Vries, S., Vohra, R.V. Combinatorial Auctions: A Survey. *INFORMS journal on Computing*, 15, 3, 2003, 284-309.

[18] Wellman, M. P., Walsh, E., Wurman, P. R. and MacKie-Mason, J. K. Auction Protocols for Decentralized Scheduling. *Games and Economic Behavior*, 35(1-2), 2001, 271-303.