Elendner, Thomas

Scheduling and combinatorial auctions: Lagrangean relaxation-based bonds for the WJISP

Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 570

Provided in Cooperation with:
Christian-Albrechts-University of Kiel, Institute of Business Administration

Suggested Citation: Elendner, Thomas (2003) : Scheduling and combinatorial auctions: Lagrangean relaxation-based bonds for the WJISP, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 570, Universität Kiel, Institut für Betriebswirtschaftslehre, Kiel

This Version is available at:
https://hdl.handle.net/10419/147634

Manuskripte
aus den
Instituten für Betriebswirtschaftslehre
der Universität Kiel

No. 570

Scheduling and Combinatorial Auctions: Lagrangean Relaxation–based Bounds for the *WJISP*

T. Elendner

Abstract

In this paper we consider the following problem: a company wants to sell consecutive time slots on a single machine. It wants to maximize the revenues, whereby market prizes are not known. Additionally, consider a number of potential buyers of those time slots providing each at least one job. All these jobs can be executed on that machine, but the number of time slots is not large enough to schedule all jobs. Furthermore assume, that every customer gains a job–specific profit with the processing of one of his jobs, whereby this profit is private information and thus, only known to him. What the company faces then is an allocation problem: Which buyer should get what time slot(s) at what prize in order to maximize the company's revenue. In the following we will show that this problem is well suited to be solved by a combinatorial auction. Within combinatorial auctions bidders are enabled to place bids on any subset of items that are auctioned off. We will focus on the winner determination problem for such an auction. The winner determination is understood as the assignment of items to bidders after all bids have been placed, with the objective to maximize the auctioneers outcome. It will be shown that the winner determination problem here can be described as so–called Weighted Job Interval Scheduling Problem ($WJISP$). Since this problem is known to be $\mathcal{NP}$–hard, heuristics have become a main research interest. Unfortunately, there has not been done much research on upper bounds so far. Furthermore, to the best of our knowledge, no runtime studies are available in the literature yet. Here, we try to close these gaps. First, we present a Lagrangean heuristic for the $WJISP$ and characterize relevant test instances. Using this test–bed, the considered upper and lower bounds are respectively compared to the LP–solution and to a heuristic taken from the literature.

Keywords: Scheduling, combinatorial auctions, winner determination, weighted job interval scheduling problem, Lagrangean relaxation, heuristics, combinatorial optimization

1 Introduction

In recent years, a new type of auction has become a main research interest, the so–called combinational or combinatorial auction.

Consider the following situation: On the one hand there is an individual who owns a number of items, whereby each item is unique and can thus be sold only once. Assume that the individual wants to maximize his revenues by selling these items but has no idea of market prizes. On the other hand think of a number of potential buyers who will gain a profit by buying these items if the prizes are below their valuations of the items. These valuations are assumed to be only known to him (private information); of course, the buyers try to minimize their expenses for these items. In addition, we presume that the buyers have superadditive preferences with respect to the items. This implies that receiving a bundle of items would have a bigger valuation to a buyer than the sum of the individual items in that bundle. For example consider a particular customer who values item $\{A\}$ at 10 currency units (cu), $\{B\}$ at 10 cu and the bundle $\{A, B\}$ at 25 cu.

An eligible allocation would assign the items in such a way to the potential buyers that the sum of the individual customers' valuations is maximized, which is called an efficient allocation. To solve this problem and to achieve an efficient allocation, different coordination concepts can be thought of, as for example bilateral negotiation or allocation rules like first come–first serve. Negotiation bears the problem of high transaction costs when negotiating with any potential buyer, because the number of potential buyers can be quite numerous in today's globalized markets; simple allocation rules on the other hand are more or less arbitrary and, thus, do

normally not yield an efficient allocation.
As mentioned above we will focus on the concept of auctions, which were identified to be well–suited in the environment of unknown market prizes and private information, see for example [30]. But applying a "traditional" like a sequential auction in the context of superadditive valuations, the customers (or bidders) would be left with a forecasting problem: assume that $\{A\}$ is auctioned off before $\{B\}$. To determine the bid for $\{A\}$ the bidder has to forecast at what prize he could get $\{B\}$ in the later auction. If, for example, he can get item $\{A\}$ at 15 cu, the bids for $\{B\}$ must not exceed 10 cu because, otherwise, he will be left with a loss.
In this context, combinatorial auctions have two complementary advantages: first, all items are auctioned off simultaneously, and second, the bidders can place bids on bundles of items and, thus, explicitly express combined values to bundles (or subsets). In the above example the bidder would be able to place three bids – on $\{A\}$, on $\{B\}$, and the bundle $\{A, B\}$ with some bidding prizes which are assumed to be smaller or equal to his valuation. In the following we will only consider bids and we will not take care if that is the real valuation or something below. Hence, mechanism design like the design of truth revelation mechanisms is beyond the scope of this paper. Also, the derivation of prizes, which is normally part of mechanism design, will not be the research topic of this paper. The readers interested in these topics are referred to [15], [19], [20], and [31], for example. Of course, the achievement of the above mentioned efficient allocation heavily depends on the truthful bidding of the participants of the auction, but here, we will assume that all bids have been placed.
Given all bids, the question for the seller (or auctioneer) reduces to the following: Which bids have to be accepted to maximize the outcome of the auction? The answer gives the so–called winner determination. It assigns the items to participants of the auction in such a way that the outcome for the auctioneer will be maximized. Given that the bids have all been placed truthfully, the winner determination will in addition achieve an efficient allocation.
This is a disadvantage of combinatorial auctions, since in the general case the winners are difficult to be obtained. The general winner determination problem can be stated as follows: given a set of T elements (items) and a set of J subsets (bids) of T, the problem is to maximize the positive–weighted subsets, whereby the subsets have to be pairwise disjoint. This denotes a combinatorial optimization problem and is proven to be $\mathcal{NP}$–hard in the general case (cp. [14]).
Despite the problems of winner determination, combinatorial auctions have recently been applied successfully in numerous ways. [22] show an application to airport time slot allocation, where the bidders have complementarities about a landing and starting slot for each aircraft. [6], [8], and [17] use the trucking services environment. These auctions have also been applied in the environment of electricity markets [21], the disposal of spectrum licenses [16], or the purchase of airtime for advertising [13], to mention only a few.
The general winner determination problem is tackled in [2], [10], [24], and [25] for example. In [23] and [28] some special cases are presented that are solvable in polynomial time. A survey on combinatorial auctions is given in [7].
In this paper, we will apply combinatorial auctions to a specific allocation problem, which will be presented in the next section. Furthermore, we derive the winner determination problem for that auction. We will then provide upper and lower bounds for that specific problem, which are essential for the successful application of exact methods. Within runtime studies we will show the quality of our bounds by comparing them to standard software and an algorithm taken from the literature, respectively.

2 An Allocation Problem

2.1 Assumptions and Notations

In this paper we study the following allocation problem: a company C has a single machine and would like to sell T contiguous time slots, $t = 1, \ldots, T$, on this machine, but it does not know any market prizes. The goal is to maximize the revenues with the sales. At the same time we have $|I|$ potential buyers, $i \in I$, of the time slots; each of them provides $|J_i|$ jobs, $j \in J_i$, and every job can be executed on this machine. In the following, the jobs will be aggregated as follows:

$$J = \bigcup_{i \in I} J_i$$

Each job $j \in J$ can be processed at most once and, once started, cannot be preempted, whereas only one job at a time can be processed. Job j can be characterized by a release date r_j, a due date d_j, and a processing time p_j. In addition, a particular customer i' gains a time–independent positive value v_j, a profit, if job j is one of his, $j \in J_{i'}$, and is scheduled on the machine. The buyers' goals are to pay as little as possible to get their jobs scheduled on the machine, or, in other words, they try to maximize their profits.
We assume that

$$\sum_{j \in J} p_j \geq T$$

and hence, not all jobs can be processed.
What is described above is a resource allocation problem. In the literature different approaches have been proposed to tackle such an allocation problem, for example, the application of multi–agent systems. There, the potential buyers (or agents) negotiate the resource among each other where each potential user tries to minimize the costs for executing his jobs. See for example [1] and the references therein.
In this paper, we will follow a decision theoretic approach similar to [29]. The question is, which customer should get what time slot(s) at what prize in order to maximize the profit of the company owning the machine. As mentioned above, prize determination is not part of the research presented here. In the following, auctions are assumed to solve the coordination problem; we will show that we face superadditive preferences within the presented allocation problem and thus, the application of combinatorial auctions becomes senseful.

2.2 Application of Combinatorial Auctions

In applying auctions to this resource allocation problem the time slots can be interpreted as items that are auctioned off by the auctioneer, i.e., company C. The jobs can be interpreted as bidders, which ask for these items. It is not essential to know, which customer provides which job, because the jobs of an individual i are not interrelated. Hence, it is possible to schedule anything between all jobs or no job for any customer $i \in I$.
Since only a time window $d_j - r_j$ for the execution of job j is given, the job can be scheduled for different consecutive time slots. These intervals can be interpreted as bids and, to be more specific, they can be interpreted as so–called XOR–bids: a job only wants to get exactly one interval of the length p_j out of interval $[r_j; d_j]$, because every job can be processed at most once. Hence, it asks for the interval

$$[r_j; r_j + p_j] \text{ XOR } [r_j + 1; r_j + p_j + 1] \text{ XOR } \ldots \text{ XOR } [d_j - p_j; d_j].$$

The bidding prize b_j for subset $j \in J_i$ is some positive value which may deviate from the true valuation v_j of customer i for this subset of time slots, with $b_j \leq v_j$.
Since the jobs can have processing times bigger than one and since they are non–preemptive, the bids for "connected" time slots can be interpreted as superadditive preferences. Applying a "traditional" auction, e.g., a sequential auction, in presence of these combinational preferences, again the already mentioned forecasting problem would evolve by auctioning one time slot at a time. Thus, we will use combinatorial auctions to solve the allocation problem where all time slots are auctioned off simultaneously and the bidders can place bids on any subset of items. Consider the following example: the quadruple $(r_{j'}, d_{j'}, p_{j'}, b_{j'}) = (0, 4, 2, 15)$ is communicated by bidder j' to the auctioneer. This means that bidder j' wants to get two consecutive time slots out of interval $[0; 4]$ and is willing to pay $b_{j'} = 15$ cu for receiving them. The bids that he places implicitly can be stated explicitly as shown in table 1.

Bid	Slot 1	Slot 2	Slot 3	Slot 4	b_j
1	1	1	0	0	15
2	0	1	1	0	15
3	0	0	1	1	15

Table 1: Combined Preferences

Any other subset of at most two time slots he values at 0 cu, because then, the job could not be processed and no profit can be gained. Note, that this are superadditive preferences; for example, his valuation of slot $\{1\}$ and $\{2\}$ is 0 each, the bundle of slots $\{1, 2\}$ he values at 15. Of course, if the bidder gets more consecutive time slots than needed, the job could also be processed. Since the additional profit for company C would be zero, C is indifferent in giving j' two or more consecutive time slots. (This situation changes if C has positive reserve prizes for each time slot. Then, bidder j' will receive at most two time slots.)

What we did up to now is to describe the allocation problem as an auction. Since we showed that the bidders (jobs) have superadditive preferences to the items (time slots), the application of combinatorial auctions becomes senseful. In the following we will present a model for the winner determination problem.

2.3 The Winner Determination Problem

Assume that all bids have been placed. Under the assumption of revenue maximization the winner determination of the mentioned combinatorial auction can be modelled as Weighted Job Interval Scheduling Problem ($WJISP$). Given $|J|$ positive–weighted jobs with job–specific release dates, due dates and processing times, the $WJISP$ assigns $|T|$ time slots in such a way to the jobs, that every time slot is assigned at most once to a job and vice versa, with the objective of maximizing the sum of the weights for the scheduled jobs. Apparently, $WJISP$ is a suitable way to model the winner determination of the presented combinatorial auction.

In this paper, we devise a time–indexed formulation of this problem similar to [11] and [27]. Let $e_j = r_j + p_j$ denote the earliest finishing times and $T'_j = [e_j; d_j]$ the possible finishing times for job j. Denoting the decision variables as

$$x_{j,t} = \begin{cases} 1, \text{ if job } j \text{ ends in time slot } t, \\ 0, \text{ otherwise,} \end{cases}$$

then, the generic $WJISP$ model can be stated as follows:

$$\max \quad \sum_{j \in J} \sum_{t=e_j}^{d_j} b_j \, x_{jt} \tag{1}$$

$$\text{s.t.} \quad \sum_{j \in J} \sum_{\tau=\max\{e_j;\, t\}}^{\min\{d_j;\, t+p_j-1\}} x_{j\tau} \leq 1 \qquad \forall t \in T \tag{2}$$

$$\sum_{t=e_j}^{d_j} x_{jt} \leq 1 \qquad \forall j \in J \tag{3}$$

$$x_{jt} \in \{0, 1\} \qquad \forall j \in J, \; \forall t \in T'_j \tag{4}$$

The objective function (1) states that the sum of the bidding prizes of the accepted bids has to be maximized. Constraints (2) state that only one job at a time is scheduled on this machine. Inequalities (3) ensure that any job is scheduled at most once and (4) denotes the domains of the decision variables.

2.4 Insights to $WJISP$

The $WJISP$ has been proven to be $\mathcal{NP}$–hard in the general case as shown in [26]. It can be reduced to the $\mathcal{NP}$–hard $JISP_k$ which is an unweighted problem where each job asks for exactly k intervals. Thus, the literature focuses on heuristic algorithms. The state of the art for heuristics can be found in [3], [4], and [5]. All three present 2–approximation algorithms for the general $WJISP$.
In [5], a deterministic two–phase algorithm (*tp* for short), consisting of a sorting, an evaluation and a selection phase, is proposed. This approach will serve as a benchmark for the heuristic derived in section 4 within computational studies. [3] apply the local–ratio technique to the $WJISP$. In [4], a rounding algorithm starting from the LP–solution is given. They also show that the LP–solution has a worst–case approximation–ratio of 2. Unfortunately, the authors do not supply any runtime studies.

Let us now analyze three polynomially solvable special cases of $WJISP$.

Theorem 1 *The special case* $\{r_j = r, \, d_j = d, \, \forall j \in J\}$ *is polynomially solvable.*

Proof. Assume the time slots on a real line. This case implies that every bidder likes to have p_j consecutive intervals whereas he is indifferent concerning the location of the time slot(s) on this line. Hence, he only has to submit the tuple (p_j, b_j). It is easy to see that this reduces to the case of combinatorial auctions with identical objects, for which the winner determination is proven to be in $\mathcal{P}$, cp. [28]. □

Theorem 2 *The special case* $\{r_j - d_j < 2p_j, \forall j \in J\}$ *is polynomially solvable.*

Proof. This is just a generalization of the case that each job is characterized by exactly one interval. The latter is proven to be polynomially solvable by dynamic programming (see [23] or [28], e.g.). Since all intervals of a job intersect, they are all mutually exclusive. □

Theorem 3 *Relaxing constraints (3) in (1) – (4) yields a polynomially solvable subproblem.*

Proof. See [23] and [28]. Again, the optimum solution can be obtained by dynamic programming. □

In the following we will calculate upper bounds by Lagrangean relaxation using the results of theorem 3.

3 Lagrangean Relaxation

In this section we are going to apply Lagrangean relaxation to (1) thru (4). Our intention is to relax the constraint set (3), which ensures that any job is scheduled at most once. Denoting the Lagrangean parameters with λ_j the relaxation can be stated as follows:

$$\max \quad \sum_{j\in J}\sum_{t=e_j}^{d_j} b_j\, x_{jt} + \sum_{j\in J}\lambda_j \left(1 - \sum_{t=e_j}^{d_j} x_{jt}\right) \tag{5}$$

$$\text{s.t.} \quad (2), \text{and } (4) \tag{6}$$

Rearranging the objective function yields

$$\max \quad \sum_{j\in J}\sum_{t=e_j}^{d_j} b'_j\, x_{jt} \left(+\sum_{j\in J}\lambda_j\right) \tag{7}$$

where

$$b'_j = b_j - \lambda_j \tag{8}$$

According to theorem 3 the subproblem is efficiently solvable by dynamic programming. The recurrence equations can be stated as follows (see [28], e.g.):

$$m(t) = \max\left\{m(t-1),\ \max\left[m(\tau) + b'_{\tau+1,t} \mid \tau < t\right]\right\} \tag{9}$$

where

$$b'_{\tau+1,t} = \max_j\{b'_{j,\tau+1,t}\} \tag{10}$$

The parameters $b'_{j,\tau+1,t}$ denote the value of job j that starts at $\tau+1$ and ends in t, reduced by λ_j. Thus, $b'_{\tau+1,t}$ gives the best bid on the interval $[\tau+1; t]$ with processing time $t-(\tau+1)$.

The value $m(T)$ gives the optimum objective function value.

To choose appropriate values for the Lagrangean multipliers λ_j we employ subgradient optimization, see for example [9] or [12]. We start with $\lambda_j = 0$ and update the multipliers after each iteration as follows: let UB and LB denote the current best known upper and lower bound, respectively, and Z the iteration counter, then

$$\lambda_j^{Z+1} = \max\left\{0; \lambda_j^Z + \theta \cdot \frac{(UB - LB) \cdot \delta_j}{\sum_{j \in J} \delta_j^2}\right\} \tag{11}$$

where

$$\delta_j = \sum_{t=e_j}^{d_j} x_{jt} - 1 \tag{12}$$

and θ denotes the step size parameter which is set to 2 initially. θ is halved, if the upper bound could not be improved within the last 5 iterations.

We implemented two stopping rules. The procedure stops, if the number of iterations reaches the limit of 250 or if the change of multipliers is sufficiently small, that is,

$$\sum_{j \in J} |\lambda_j^{Z+1} - \lambda_j^Z| < \epsilon \tag{13}$$

where we chose $\epsilon = 0.001$.

An essential result of duality theory states that the Lagrangean relaxation (7), (2), and (4) achieves at least the objective function value of the LP–relaxation of (1) – (4). Furthermore, in the case that the integrality property holds the objective function value of the Lagrangean relaxation equals that of the LP–relaxation.

Theorem 4 *For the subproblem (7), (2), and (4), the integrality property holds.*

Proof. The condition that any job can be processed only once having been relaxed, it is easy to see that the remaining problem has the consecutive ones property. For that case, all extreme points of the underlying polyhedron are integral, see for example [18]. □

Hence, the best upper bound that we can achieve by the assumed Lagrangean relaxation is the objective function value of the LP–relaxation. Nevertheless, it is interesting to see the runtime performance of the Lagrangean relaxation in comparison to the LP–relaxation.

4 Lagrangean Heuristic

In this section we develop a heuristic which uses the results given by the Lagrangean relaxation. Recall, that by the chosen relaxation it is possible to schedule jobs more often than once. The first step of the algorithm is to construct a feasible solution to $WJISP$ from the solution of (9). Any job which has been chosen at least once by the Lagrangean relaxation is considered

to be in the initial solution. For jobs scheduled at least twice, we build a feasible schedule by randomly choosing one occurrence of this job. Since this job has already been scheduled, we reduce the number of schedulable jobs and update the available time slots.
A detailed description is given within algorithm 1.

Algorithm 1 *(Initial solution)*

Input: solution $\mathbb{L}$ *of (9),* T, J, $(r_j, d_j, p_j, b_j)\ \forall j \in J$;
Output: feasible solution $\mathbb{L}'$, LB, *flag,* T', J';
$T' \leftarrow T$, $J' \leftarrow J$;
flag $= 0$;
for $(j \in \mathbb{L})$ {
if $(\sum_t x_{j't} \geq 1)$ {
if $(\sum_t x_{j't} \geq 2)$ {
flag $= 1$;
choose t' *with* $x_{j't'} = 1$ *at random;*
}
$\mathbb{L}' \leftarrow \mathbb{L}' \cup \{x_{j't'}\}$;
$LB = LB + b_{j'}$;
$J' \leftarrow J' \backslash \{j'\}$;
$T' \leftarrow T' \backslash \{t', t'-1, \ldots, t'-p_{j'}+1\}$;
}
}

The result of algorithm 1 is a feasible solution to $WJISP$. If it returns *flag* $= 0$, we know that (9) already produced a feasible solution.
Note that at least one time slot will be empty, if *flag* $= 1$ is returned. In that case, the improvement phase takes over. There, we consider those jobs and time slots that have not been scheduled and occupied yet, respectively. If we imagine the time slots being on a real line, and if you remove the time slots already occupied, the real line will become clustered. The improvement algorithm solves (9) separately on any remaining consecutive time interval on this line. Of course, only unscheduled jobs are taken into account.
Algorithm 2 gives a description of the improvement phase.
It is possible that the improvement phase yields again a plan where jobs are scheduled at least twice. Therefore, the solution of this problem is again transferred to algorithm 1 and so on. This iterative procedure stops if the parameter *flag* returns 0 and we get a feasible solution to $WJISP$. The heuristic is solved after each iteration of the Lagrangean relaxation.

Algorithm 2 *(Improvement)*

Input: flag, T', J';
Output: solution $\mathbb{L}$ of (9), T, J;
if (*flag* $== 1$) {
  for (*all intervals in* T') {
    solve (9) on J';
    obtain new solution $\mathbb{L}$;
  }
}
else stop
$T \leftarrow T'$, $J \leftarrow J'$;

5 Computational Studies

In order to test the performance of the algorithm, we implemented the Lagrangean relaxation and heuristic (lh for short) in GNU C. The relaxation is tested against the LP–relaxation (lp for short) of (1)–(4), which is solved by CPLEX 7.0. As mentioned above, the lower bound is benchmarked against the two–phase algorithm (tp) proposed in [5] which was implemented in GNU C as well.
The tests are being performed on an AMD Athlon with a 1.8 GHz processor and 768 MB of RAM running a Linux operating system.

5.1 Test Instances

To the best of our knowledge there have not been any runtime studies on algorithms for the $WJISP$ in the literature. To close this gap, first we develop an instance generator. The basic idea stems from an instance generator for general combinatorial auctions which can be found in [24].
For generating an instance the following parameters need to be defined:

J	:	number of jobs
T	:	number of time slots
l^p, u^p	:	lower and upper bound for processing time
l^b, u^b	:	lower and upper bound for bidding prize
ϕ	:	distribution for bidding prizes

Every particular job j' is assigned a processing time which is chosen out of the interval $[l^p; u^p]$. Then, the release date is determined out of the feasible interval $[0; T - p_{j'}]$. After that the due date is generated out of $[r_{j'} + p_{j'}; T]$.
All data are generated randomly and uniformly distributed on the particular intervals. Without loss of generality we assume all the data to be integer–valued.
We choose two different distributions $\phi \in \Phi$ for the bidding prizes, random (ra) and weighted random (wra). In both cases at first we draw a random number $r_{j'}$ uniformly distributed in the interval $[l^b; u^b]$. For $\phi = ra$, we directly assign the value as bidding prize, thus, $b_{j'} = r_{j'}$. Under the assumption of wra we multiply the random number with the processing time, i. e., we calculate $b_{j'} = r_{j'} p_{j'}$. The intention is that the more time slots are requested in a specific

bid the higher the bidding prize tends to be. This latter assumption seems to be more realistic. The jobs are stored as quadruples as already mentioned. Thus, any instance consists of a value for T and J, and $|J|$ quadruples, one for each generated job.

According to the results of pretests and in order to give an extensive test–bed, we chose the number of jobs and time slots as follows:

$$J \in \{200, 400, 600, 800, 1000\}$$

$$T \in \{100, 200, 400, 600, 800, 1000\}$$

with the assumption of $J \geq T$. The number of possible combinations are

$$2 + 3 + 4 + 5 + 6 = 20.$$

Furthermore, we chose the possible intervals for the processing times $[l^p; u^p]$ out of the set I, where

$$I = \left\{ \left[1; \frac{T}{4}\right], \left[1; \frac{T}{2}\right], \left[1; \frac{3T}{4}\right], [1; T], \left[\frac{T}{4}; \frac{3T}{4}\right] \right\}.$$

Finally we picked an interval for the bidding prizes:

$$[l^b; u^b] = [1000; 10000]$$

Hence, we did not test the influence of different intervals for bidding prizes.
Since we generated 5 instances each for these parameter settings with $|\Phi| = 2$ distributions, in total we get

$$20 \cdot 5 \cdot 1 \cdot 5 \cdot 2 = 1000$$

instances.

5.2 Results

Any instance can be characterised by the triple $\{\gamma \in \Gamma; \phi \in \Phi; i \in I\}$. Here, γ is expressed as jo_ts, which denotes the number of jobs and time–slots, each divided by 100. Parameter $\phi \in \Phi$ is expressed as $\{ra, wra\}$ and therefore denotes the distribution of the bidding prizes, and $i \in I$ represents the interval, from which the processing times of the jobs are drawn.
Since the calculation of our lower bound contains a probabilistic component, the instances were all executed twice for the Lagrangean heuristic. We observed that for ra (wra) the deviation of the two runs for the lower bound is about 7% (2%) for the worst and 1% (below 0.01%) for the average case over all double–runs. The deviation of the upper bounds were in the worst (average) case below 3% (0.1%) for both distributions. Because the solution for tp and the derivation of the LP–solution are deterministic, those algorithms were only executed once.

In order to give an idea of how far standard software – in our case CPLEX 7.0 – can reach for the chosen test environment, we at first applied CPLEX to derive the solution of the LP-relaxation for the instances. Table 2 shows how many instances could be solved. Although

γ	ϕ / i	ra					wra				
		$[1;\frac{T}{4}]$	$[1;\frac{T}{2}]$	$[1;\frac{3T}{4}]$	$[1;T]$	$[\frac{T}{4};\frac{3T}{4}]$	$[1;\frac{T}{4}]$	$[1;\frac{T}{2}]$	$[1;\frac{3T}{4}]$	$[1;T]$	$[\frac{T}{4};\frac{3T}{4}]$
2_1		5	5	5	5	5	5	5	5	5	5
2_2		5	5	5	5	5	5	5	5	5	5
4_1		5	5	5	5	5	5	5	5	5	5
4_2		5	5	5	5	5	5	5	5	5	5
4_4		5	5	5	5	5	5	5	5	5	5
6_1		5	5	5	5	5	5	5	5	5	5
6_2		5	5	5	5	5	5	5	5	5	5
6_4		5	5	5	5	5	5	5	5	5	5
6_6		5	5	5	4	–	5	5	5	5	–
8_1		5	5	5	5	5	5	5	5	5	5
8_2		5	5	5	5	5	5	5	5	5	5
8_4		5	5	5	5	5	5	5	5	5	5
8_6		5	–	–	–	–	5	–	–	–	–
8_8		–	–	–	–	–	–	–	–	–	–
10_1		5	5	5	5	5	5	5	5	5	5
10_2		5	5	5	5	5	5	5	5	5	5
10_4		5	5	4	5	5	5	5	5	5	5
10_6		5	–	–	–	–	5	–	–	–	–
10_8		–	–	–	–	–	–	–	–	–	–
10_10		–	–	–	–	–	–	–	–	–	–
Σ		85	75	74	74	70	85	75	75	75	70

Table 2: Number of Solved Instances by CPLEX 7.0

providing 768 MB of RAM, not all instances could be solved. Here, entry "–" indicates that no instance could be solved. It can be observed, that the distribution ϕ does not influence results significantly. The only thing that can be stated is that $\left\{.;.;\left[1;\frac{T}{4}\right]\right\}$ could be solved best and $\left\{.;.;\left[\frac{T}{4};\frac{3T}{4}\right]\right\}$ worst. In contrast, the Lagrangean heuristic and the two–phase algorithm both solved all instances.

Tables 3 and 4 give information about the runtimes of lp and lh within the specific distributions. min, max, and avg denote the absolute minimum, absolute maximum and average over all instances within a particular column. Hence, the worst running time of lp over all 100 instances of $\left\{\gamma \in \Gamma; ra; \left[1;\frac{T}{4}\right]\right\}$ was 39 seconds. An entry "**" indicates that none of the instances could be solved. Thus, runtimes for that cases could not be suggested for calculation of min, max, and avg. Nevertheless, it is to see that throughout the instances the runtimes for lp are significantly higher.

Focussing on lh, we can observe a slight decrease of the runtimes by increasing mean of processing times of the jobs. Since a probabilistic component and two different stopping rules are included, it is not stringent. Looking at the distributions of the bids, we can see that the running times for wra are significantly lower than for ra. The reason is that for ra normally all 250 iterations had to be executed; for wra, the stopping criterion in (13) was reached often in early iterations. An outlier for wra was an instance $\{10\_10; wra; [1;T]\}$ with about 13 seconds solution time. Here, all 250 iterations had to be executed.

Next, we study the quality of our upper bound. To do so we compare our upper bound to the best bound we can achieve with lh, i.e. the solution of lp. As a measure we use the following expression for this comparison:

i	$[1;\frac{T}{4}]$		$[1;\frac{T}{2}]$		$[1;\frac{3T}{4}]$		$[1;T]$		$[\frac{T}{4};\frac{3T}{4}]$	
γ	lp	lh	lp	lh	lp	lh	lp	lh	lp	lh
2_1	0.12	0.10	0.13	0.09	0.16	0.08	0.17	0.08	0.14	0.00
2_2	0.42	0.37	0.76	0.36	0.97	0.35	0.85	0.32	0.64	0.00
4_1	0.28	0.15	0.41	0.14	0.44	0.12	0.41	0.12	0.39	0.02
4_2	1.07	0.46	1.56	0.44	1.56	0.43	1.94	0.40	1.52	0.00
4_4	6.22	2.03	8.12	2.21	8.46	2.23	7.26	2.14	6.26	0.04
6_1	0.40	0.21	0.64	0.18	0.79	0.17	0.62	0.16	0.68	0.00
6_2	1.64	0.55	2.20	0.55	2.56	0.48	2.26	0.46	2.36	0.01
6_4	8.00	2.24	9.36	2.54	10.94	2.54	10.86	2.39	10.46	0.03
6_6	20.80	5.63	25.60	5.84	32.20	5.63	28.00	5.18	**	0.08
8_1	0.61	0.27	0.94	0.24	0.95	0.22	0.88	0.20	0.93	0.01
8_2	2.30	0.66	3.32	0.60	3.10	0.57	3.30	0.52	3.38	0.02
8_4	12.00	2.40	14.60	2.77	13.80	2.85	14.20	2.68	12.40	0.06
8_6	30.40	6.29	**	6.49	**	6.22	**	5.83	**	0.14
8_8	**	10.70	**	10.51	**	9.93	**	9.26	**	0.24
10_1	0.79	0.34	1.02	0.30	1.16	0.27	1.05	0.24	1.04	0.01
10_2	2.92	0.77	3.88	0.69	4.00	0.67	4.08	0.59	4.04	0.03
10_4	15.00	2.57	19.60	3.06	17.75	3.14	17.60	3.00	16.40	0.13
10_6	34.40	6.90	**	7.20	**	6.88	**	6.36	**	0.33
10_8	**	11.62	**	11.58	**	10.91	**	9.96	**	0.59
10_10	**	16.31	**	15.72	**	14.89	**	13.67	**	0.81
min	0.10	0.08	0.10	0.08	0.12	0.07	0.13	0.07	0.12	0.00
avg	8.08	1.60	6.14	1.00	6.44	0.99	5.94	0.92	4.33	0.02
max	39.00	7.01	28.00	6.08	38.00	5.79	32.00	5.30	21.00	0.62

Table 3: Average Runtimes for lp and lh, Distribution ra

i	$[1;\frac{T}{4}]$		$[1;\frac{T}{2}]$		$[1;\frac{3T}{4}]$		$[1;T]$		$[\frac{T}{4};\frac{3T}{4}]$	
γ	lp	lh	lp	lh	lp	lh	lp	lh	lp	lh
2_1	0.15	0.03	0.20	0.02	0.19	0.01	0.16	0.01	0.15	0.00
2_2	0.58	0.13	0.83	0.14	1.11	0.10	0.78	0.07	0.69	0.00
4_1	0.32	0.03	0.50	0.04	0.57	0.00	0.51	0.00	0.43	0.00
4_2	1.28	0.16	1.60	0.07	1.82	0.18	2.00	0.09	1.66	0.00
4_4	7.64	0.97	8.58	0.46	9.18	1.07	6.34	0.22	6.50	0.01
6_1	0.62	0.04	0.77	0.02	0.80	0.02	0.72	0.01	0.65	0.00
6_2	2.24	0.14	2.24	0.09	2.38	0.16	2.12	0.14	2.24	0.00
6_4	8.08	0.71	11.10	0.61	11.80	0.66	10.96	0.54	9.88	0.01
6_6	27.80	2.03	35.60	1.14	26.40	1.46	26.00	2.29	**	0.02
8_1	0.67	0.05	1.12	0.02	1.12	0.02	1.01	0.02	0.97	0.00
8_2	3.16	0.20	3.52	0.05	4.02	0.15	3.40	0.11	3.08	0.00
8_4	16.20	0.70	16.60	0.65	18.00	1.00	15.00	0.64	13.00	0.01
8_6	36.60	1.92	**	1.76	**	2.75	**	2.50	**	0.02
8_8	**	4.93	**	3.18	**	1.30	**	4.56	**	0.03
10_1	0.90	0.09	1.38	0.03	1.62	0.04	1.32	0.05	1.20	0.00
10_2	5.00	0.15	4.28	0.07	5.10	0.14	4.32	0.11	3.96	0.00
10_4	19.80	0.72	20.80	0.87	23.80	1.06	20.80	1.12	17.00	0.01
10_6	49.60	2.17	**	1.52	**	1.88	**	2.06	**	0.02
10_8	**	3.63	**	2.78	**	1.33	**	2.68	**	0.04
10_10	**	5.57	**	4.09	**	3.72	**	6.15	**	0.05
min	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
avg	10.78	1.22	7.28	0.88	7.19	0.85	6.36	1.17	4.39	0.01
max	57.00	10.06	45.00	5.77	29.00	6.07	28.00	12.97	18.00	0.06

Table 4: Average Runtimes for lp and lh, Distribution wra

γ	ϕ / i	ra					wra				
		$[1;\frac{T}{4}]$	$[1;\frac{T}{2}]$	$[1;\frac{3T}{4}]$	$[1;T]$	$[\frac{T}{4};\frac{3T}{4}]$	$[1;\frac{T}{4}]$	$[1;\frac{T}{2}]$	$[1;\frac{3T}{4}]$	$[1;T]$	$[\frac{T}{4};\frac{3T}{4}]$
2_1		0.08	0.08	0.11	0.08	0.00	1.19	0.20	0.20	0.00	0.00
2_2		0.04	0.57	0.43	0.01	0.00	1.18	0.23	0.27	0.04	0.00
4_1		0.31	1.28	0.49	0.97	0.03	0.43	0.07	0.00	0.01	0.00
4_2		0.13	0.39	0.35	0.87	0.00	0.44	0.25	0.08	0.03	0.00
4_4		0.07	0.57	0.45	1.23	0.00	0.47	0.16	0.14	0.01	0.00
6_1		0.42	1.18	0.87	0.73	0.00	0.38	0.09	0.10	0.23	0.00
6_2		0.49	0.74	0.74	1.18	0.00	0.39	0.07	0.13	0.28	0.00
6_4		0.27	0.28	0.89	0.57	0.00	0.38	0.09	0.24	0.25	0.00
6_6		0.23	0.67	0.60	0.35	**	0.41	0.10	0.23	0.26	**
8_1		0.33	0.65	0.88	1.81	0.00	0.16	0.07	0.10	0.14	0.00
8_2		0.45	0.84	1.31	1.65	0.00	0.17	0.06	0.06	0.14	0.00
8_4		0.31	0.37	1.39	1.15	0.00	0.18	0.04	0.09	0.26	0.00
8_6		0.16	**	**	**	**	0.19	**	**	**	**
8_8		**	**	**	**	**	**	**	**	**	**
10_1		0.41	0.50	1.16	1.69	0.00	0.09	0.06	0.03	0.10	0.00
10_2		0.41	0.95	1.54	2.34	0.00	0.14	0.06	0.04	0.14	0.00
10_4		0.16	0.73	1.26	1.85	0.00	0.11	0.04	0.04	0.16	0.00
10_6		0.24	**	**	**	**	0.12	**	**	**	**
10_8		**	**	**	**	**	**	**	**	**	**
10_10		**	**	**	**	**	**	**	**	**	**
min		0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
avg		0.23	0.49	0.62	0.82	0.00	0.32	0.08	0.09	0.10	0.00
max		1.27	2.25	2.15	3.48	0.14	2.60	0.93	0.79	0.97	0.00

Table 5: Average, Best, and Worst Performance of lh compared to lp

$$\frac{(UB_{lh} - UB_{lp})}{UB_{lh}} \cdot 100\,\%. \tag{14}$$

UB_{lh} and UB_{lp} respectively denotes the upper bound for the Lagrangean heuristic and the LP–relaxation. UB_{lh} gives the average over the double–runs. Thus, (14) be interpreted as the deviation of our upper bound to the LP-solution, expressed in percent. In table 5, the observed deviations are listed, respectively averaged over the number of instances which could be solved by CPLEX. It can be stated that our upper bound is very tight in comparison to the solution of lp. It ranges from 0.0 to 3.5% in the worst cases and is below 1% on each average of $\{.; \phi \in \Phi; i \in I\}$. In addition, we can see that wra could normally be solved far better than ra using lh.

Now we analyze the quality of our lower bounds and we use tp as a benchmark. For the sake of completeness, we start with providing some implementation details and running times of tp. In tp, a sorting algorithm is needed. To this, quicksort was used. As a rule of thumb we can state that the running time for tp consists at least half of sorting time. Furthermore, the algorithm needs a different representation of the instances; whereas for lh the aforementioned implicit bids (quadruples) could be used, tp needs every single interval, and hence, the explicit representation of bids as demonstrated in table 1. The times for deriving these intervals were not taken into consideration.
Table 6 shows the average running times for tp. It becomes obvious, that the running times for tp are significantly lower than for lh. But, since lh also delivers an upper bound, these times cannot be compared. For tp, there is no big difference in respect of running times for ra

γ	ϕ / i	ra $[1;\frac{T}{4}]$	ra $[1;\frac{T}{2}]$	ra $[1;\frac{3T}{4}]$	ra $[1;T]$	ra $[\frac{T}{4};\frac{3T}{4}]$	wra $[1;\frac{T}{4}]$	wra $[1;\frac{T}{2}]$	wra $[1;\frac{3T}{4}]$	wra $[1;T]$	wra $[\frac{T}{4};\frac{3T}{4}]$
2_1		0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00
2_2		0.01	0.00	0.01	0.01	0.00	0.01	0.01	0.01	0.00	0.00
4_1		0.01	0.01	0.01	0.00	0.00	0.02	0.01	0.01	0.01	0.00
4_2		0.04	0.03	0.02	0.01	0.01	0.04	0.03	0.02	0.01	0.01
4_4		0.08	0.06	0.04	0.03	0.02	0.08	0.06	0.05	0.03	0.02
6_1		0.03	0.02	0.02	0.01	0.01	0.03	0.02	0.02	0.01	0.01
6_2		0.07	0.05	0.04	0.02	0.02	0.08	0.06	0.04	0.03	0.02
6_4		0.16	0.12	0.09	0.06	0.04	0.18	0.12	0.10	0.06	0.04
6_6		0.27	0.19	0.13	0.09	0.07	0.29	0.19	0.15	0.10	0.06
8_1		0.06	0.04	0.03	0.02	0.02	0.05	0.04	0.03	0.02	0.02
8_2		0.12	0.09	0.06	0.04	0.03	0.13	0.10	0.07	0.05	0.03
8_4		0.29	0.21	0.15	0.09	0.07	0.32	0.22	0.15	0.10	0.07
8_6		0.49	0.34	0.23	0.16	0.12	0.56	0.36	0.25	0.16	0.12
8_8		0.74	0.47	0.33	0.21	0.18	0.81	0.52	0.36	0.24	0.18
10_1		0.08	0.06	0.04	0.03	0.02	0.09	0.07	0.04	0.03	0.03
10_2		0.19	0.13	0.09	0.07	0.05	0.21	0.15	0.10	0.07	0.06
10_4		0.48	0.32	0.22	0.15	0.12	0.54	0.37	0.24	0.16	0.12
10_6		0.83	0.55	0.37	0.23	0.21	0.90	0.64	0.41	0.26	0.21
10_8		1.21	0.80	0.53	0.34	0.32	1.35	0.91	0.58	0.37	0.31
10_10		1.63	1.07	0.72	0.44	0.44	1.82	1.27	0.80	0.48	0.41
min		0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
avg		0.34	0.23	0.16	0.10	0.09	0.38	0.26	0.17	0.11	0.09
max		1.78	1.14	0.76	0.46	0.54	2.12	1.36	0.85	0.53	0.44

Table 6: Average Runtimes for tp

and wra, since for this algorithm the number of intervals heavily influences the running times. With respect to the influence of set I it can be stated, that the runtime of tp decreases with increasing mean of the intervals for the processing times.
Focussing the solution quality of lh in comparison to tp on average and in the worst case, we use

$$\frac{(LB_{lh} - LB_{tp})}{LB_{lh}} \cdot 100\,\%. \tag{15}$$

Here, LB_{tp} denotes the achieved lower bound by algorithm tp. LB_{lh} is the weighted average over the double–runs for lh. Thus, expression (15) can be interpreted as the percentage improvement of LB_{lh} to LB_{tp}. Hence, a positive value implies that on average the solution of LB_{lh} was better than that of LB_{tp}. Table 7 shows the observed values.
It can be stated that in general the lower bound derived by lh is on average better than that by tp. Only in 7 out of 1000 instances tp performs better than lh.
Interestingly, all these 7 cases appear under $\phi = ra$ and $i = \left[1;\frac{T}{4}\right]$. Nevertheless, also for these instances lh gives better bounds on average.
The deviation from LB_{lh} to LB_{tp} tends to decrease when the instance size increases. Furthermore, it slightly increases with increasing mean of processing times of the jobs, but this effect decreases when the instance size grows. An obvious outlier is the case $\phi = ra$ and $i = \left[\frac{T}{4};\frac{3T}{4}\right]$. These instances could be solved by tp quite good as well.
Looking at the overall averages, the increase of the averages becomes more stringent, with the same outlier. These averages are positive throughout the instances. They are at least above 6% and at most around 16%. The best cases for lh range between 17% and 35%, the worst

γ	ϕ / i	ra $[1;\frac{T}{4}]$	ra $[1;\frac{T}{2}]$	ra $[1;\frac{3T}{4}]$	ra $[1;T]$	ra $[\frac{T}{4};\frac{3T}{4}]$	wra $[1;\frac{T}{4}]$	wra $[1;\frac{T}{2}]$	wra $[1;\frac{3T}{4}]$	wra $[1;T]$	wra $[\frac{T}{4};\frac{3T}{4}]$
2_1		9.25	14.80	19.08	15.92	14.78	12.75	12.40	14.12	14.75	15.82
2_2		13.31	13.09	19.29	20.33	13.69	10.86	9.26	12.63	15.69	16.16
4_1		7.50	12.27	14.29	18.27	9.68	11.83	15.70	18.42	14.40	18.35
4_2		12.32	14.14	17.65	16.13	14.25	11.07	13.66	14.97	17.86	19.89
4_4		11.24	14.51	15.57	21.00	7.45	12.18	15.63	15.60	16.44	16.79
6_1		6.93	14.56	12.02	13.09	9.85	12.11	13.42	14.99	13.51	15.64
6_2		5.98	11.81	15.52	14.40	10.03	11.14	14.38	14.47	14.82	19.60
6_4		5.65	12.29	16.14	15.24	7.83	12.30	13.69	17.49	16.79	14.61
6_6		8.02	11.93	14.32	16.13	8.57	10.77	15.86	16.92	16.73	11.97
8_1		6.30	12.43	12.12	13.15	9.13	10.27	13.09	15.42	12.60	17.95
8_2		3.71	14.68	13.52	13.23	11.80	11.94	14.28	14.76	14.61	19.02
8_4		5.07	14.45	12.10	11.41	9.81	12.40	14.78	13.43	16.72	14.52
8_6		3.61	16.70	12.73	13.05	9.92	12.41	14.75	14.86	17.45	14.47
8_8		4.24	14.86	12.27	12.45	9.62	10.28	14.07	14.86	16.45	16.41
10_1		3.99	7.71	9.25	11.26	12.63	11.74	12.90	12.14	12.53	16.93
10_2		3.47	9.14	11.00	11.98	8.31	13.25	14.16	12.07	14.10	17.12
10_4		6.26	11.02	9.49	10.69	10.99	13.19	14.15	12.52	13.38	9.05
10_6		2.94	10.40	12.54	12.41	9.35	13.75	13.89	12.72	12.25	16.16
10_8		2.87	10.66	12.77	12.06	10.18	13.10	14.16	14.84	16.14	15.31
10_10		1.99	10.89	11.45	13.29	8.37	13.01	13.09	12.40	16.49	13.28
min		-5.05	4.95	3.29	5.85	0.00	4.49	4.76	5.70	5.79	0.00
avg		6.23	12.62	13.66	14.27	10.31	12.02	13.87	14.48	15.19	15.95
max		19.92	19.76	30.75	31.59	31.00	17.86	24.20	26.54	28.29	35.50

Table 7: Average, Best, and Worst Performance of lh compared to tp

between -5% and 6%.
In our opinion the most "realistic" instances for our allocation problem is the case $\phi = wra$ and $i = \left[1;\frac{T}{4}\right]$. For these instances the mean deviation for any instance size is about 12% and above 4% in the worst case.

Up to now we focussed the quality for our lower and upper bounds separately. To give an idea how far upper and lower bound are apart we will last but not least analyze the deviation of these bounds for lh. The quality will be measured by the following expression:

$$\frac{(UB_{lh} - LB_{lh})}{UB_{lh}} \cdot 100\,\%. \tag{16}$$

UB_{lh} and LB_{lh} denote the best upper and lower bound that could be found by executing lh. Hence, it is the average percentage deviation from upper to lower bound. The deviations can be found in table 8, averaged over all 10 runs for each entry $\{\gamma \in \Gamma; \phi \in \Phi; i \in I\}$.
It becomes obvious, that the increase of the mean of processing times of the jobs is accompanied by a slight decrease of the deviation. For $\phi = ra$, the increase of the instance size tends to lead to bigger deviations of upper and lower bounds. Surprisingly, for $\phi = wra$ we can state the contrary observation. Again, the stated propositions do not hold stringently. The worst average deviation was reached by the instances of $\left(10\_8; ra; \left[1;\frac{T}{4}\right]\right)$ with below 20% in mean.
Focussing on ϕ it can be stated that the quality for wra is significantly better than that for ra. Note, that all instances $\left\{\gamma; wra; \left[\frac{T}{4};\frac{3T}{4}\right]\right\}$ were solved to optimality.

γ	ϕ / i	ra $[1;\frac{T}{4}]$	$[1;\frac{T}{2}]$	$[1;\frac{3T}{4}]$	$[1;T]$	$[\frac{T}{4};\frac{3T}{4}]$	wra $[1;\frac{T}{4}]$	$[1;\frac{T}{2}]$	$[1;\frac{3T}{4}]$	$[1;T]$	$[\frac{T}{4};\frac{3T}{4}]$
2_1		8.83	4.72	3.76	5.49	0.00	2.41	0.71	0.36	0.13	0.00
2_2		9.17	6.39	3.89	4.76	0.00	2.12	0.75	0.79	0.19	0.00
4_1		8.87	7.60	7.17	4.38	0.13	0.81	0.71	0.00	0.02	0.00
4_2		7.96	7.18	8.66	6.50	0.20	0.83	0.50	0.56	0.06	0.00
4_4		9.88	9.24	8.88	7.11	0.19	1.36	0.35	0.66	0.01	0.00
6_1		10.28	9.22	8.75	8.63	0.10	0.91	0.19	0.33	0.43	0.00
6_2		13.45	12.37	8.55	10.06	0.10	0.74	0.18	0.36	0.55	0.00
6_4		15.29	12.80	10.33	8.58	0.10	0.56	0.22	0.51	0.30	0.00
6_6		14.91	12.73	11.68	8.36	0.10	1.03	0.20	0.48	0.45	0.00
8_1		10.28	9.02	8.12	9.83	0.10	0.33	0.18	0.19	0.40	0.00
8_2		14.81	10.72	9.41	11.35	0.10	0.48	0.14	0.20	0.44	0.00
8_4		15.76	11.38	12.48	11.49	0.10	0.36	0.08	0.24	0.40	0.00
8_6		17.66	11.18	11.08	10.98	0.10	0.39	0.13	0.32	0.33	0.00
8_8		16.80	11.62	12.89	10.19	0.10	0.40	0.24	0.23	0.64	0.00
10_1		10.01	10.92	10.97	8.01	0.10	0.24	0.17	0.10	0.41	0.00
10_2		16.25	13.14	12.51	10.22	0.10	0.38	0.12	0.24	0.25	0.00
10_4		17.66	12.09	14.16	11.85	0.10	0.26	0.31	0.18	0.65	0.00
10_6		18.79	12.26	13.35	11.81	0.10	0.36	0.15	0.12	0.82	0.00
10_8		19.35	13.16	14.00	10.85	0.10	0.33	0.19	0.06	0.44	0.00
10_10		19.28	13.06	14.52	10.75	0.10	0.36	0.27	0.24	0.36	0.00
min		3.39	1.14	1.74	1.46	0.00	0.01	0.00	0.00	0.00	0.00
avg		13.76	10.54	10.26	9.06	0.10	0.73	0.29	0.31	0.36	0.00
max		22.58	18.55	19.27	18.64	0.51	4.02	2.36	1.97	1.92	0.00

Table 8: Average Quality for lh

Looking at the observed overall worst cases we can state that it ranges between 0 and 22.5% for ra and 0 and 4% for wra.

6 Conclusions

In this paper we focus on a specific allocation problem which is solved by a combinatorial auction. We consider the winner determination problem and show that it can be modelled as $WJISP$. Thus, a new application for $WJISP$ is pointed out.
We show that the standard software package CPLEX is overstrained solving the LP–relaxation of $WJISP$ for larger instances. Hence, new upper and lower bounds based on Lagrangean relaxation are derived. We compare the performance of our bounds to the LP–relaxation and a heuristic taken from the literature. Therefore, a new instance generator is provided. It becomes obvious that our upper bound is tight in comparison to the LP-solution and that our lower bound outperforms that of the algorithm taken from the literature almost always.

Acknowledgement: We like to thank Andreas Drexl for his continuous support, Birgitta Elendner, Eike Houben, Alf Kimms, Sonja Kovaleva, and Rudolf Müller for their helpful hints, and Ingo Strunk, who took over parts of the programming.

References

[1] A. AGNETIS, P. B. MIRCHANDANI, D. PACCIARELLI, AND A. PACIFICI, *Scheduling problems with two competing users*, tech. report, Università Roma Tre, Dipartimento di Informatica e Automazione, Rome, 2001.

[2] A. ANDERSSON, M. TENHUNEN, AND F. YGGE, *Integer Programming for Combinatorial Auction Winner Determination*, tech. report, Computer Science Department, Uppsala University, Uppsala, 2000.

[3] A. BAR-NOY, R. BAR-YEHUDA, A. FREUND, J. NAOR, AND B. SCHIEBER, *A unified approach to approximating resource allocation and scheduling*, Journal of the ACM, 48 (2001), pp. 1069–1090.

[4] A. BAR-NOY, S. GUHA, J. NAOR, AND B. SCHIEBER, *Approximating the throughput of multiple machines in real-time scheduling*, SIAM Journal on Computing, 31 (2001), pp. 331–352.

[5] P. BERMAN AND B. DASGUPTA, *Multi-phase algorithms for throughput maximization for real-time scheduling*, Journal of Combinatorial Optimization, 4 (2000), pp. 307–323.

[6] C. G. CAPLICE, *An Optimization Based Bidding Process: A New Framework for Shipper-Carrier Relationships*, PhD thesis, Massachusetts Institute of Technology, 1996.

[7] S. DE VRIES AND R. VOHRA, *Combinatorial Auctions: A Survey*, INFORMS Journal on Computing, 15 (2003). to appear.

[8] T. ELENDNER, B. BURMEISTER, AND T. IHDE, *How Combinatorial Auctions work for DaimlerChrysler*. in preparation.

[9] M. L. FISHER, *The Lagrangian Relaxation Method for Solving Integer Programming Problems*, Management Science, 27 (1981), pp. 1–18.

[10] Y. FUJISHIMA, K. LEYTON-BROWN, AND Y. SHOHAM, *Taming the Computational Complexity of Combinatorial Auctions: Optimal and Approximate Approaches*, tech. report, Computer Science Department, Stanford University, Stanford, 1999.

[11] L. A. HALL, D. B. SHMOYS, AND J. WEIN, *Scheduling to minimize average completion time: Off-line and on-line algorithms*, in Proceeding of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms, 1996, pp. 142–151.

[12] M. H. HELD, P. WOLFE, AND H. D. CROWDER, *Validation of subgradient optimization*, Mathematical Programming, 6 (1975), pp. 62–88.

[13] J. L. JONES, *Incompletely Specified Combinatorial Auctions: An Alternative Allocation Mechanism for Business-to-Business Negotiations*, PhD thesis, University of Florida, 2000.

[14] R. M. KARP, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., 1972, pp. 85–103.

[15] J. K. MACKIE-MASON AND H. R. VARIAN, *Generalized Vickrey Auctions*, tech. report, University of Michigan, Department of Economics, July 1995.

[16] J. McMillan, *Selling Spectrum Rights*, Journal of Economic Perspectives, 8 (1994), pp. 145–162.

[17] E. W. Moore, J. M. Warmke, and L. R. Gorban, *The Indispensable Role of Management Science in Centralizing Freight Operations at Reynolds Metals Company*, Interfaces, 21 (1991), pp. 107–129.

[18] G. L. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization*, New York, 1988.

[19] D. C. Parkes, *Iterative Combinatorial Auctions: Achieving Economic and Computational Efficiency*, PhD thesis, University of Pennsylvania, 2001.

[20] D. C. Parkes and L. H. Ungar, *Iterative Combinatorial Auctions: Theory and Practice*, in Proceedings of the 17th National Conference on Artificial Intelligence (AAAI-00), 2000, pp. 74–81.

[21] J. E. Quintero, *Combinatorial Electricity Auctions*, tech. report, Management Science and Engineering Department, Stanford University, Stanford, 2000.

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

[23] M. H. Rothkopf, A. Pekeč, and R. M. Harstad, *Computationally Manageable Combinational Auctions*, Management Science, 44 (1998), pp. 1131–1147.

[24] T. Sandholm, *Algorithm for optimal winner determination in combinatorial auctions*, Artificial Intelligence, 135 (2002), pp. 1–54.

[25] T. Sandholm and S. Suri, *Improved Algorithms for Optimal Winner Determination in Combinatorial Auctions and Generalizations*, in Proceedings of the National Conference on Artificial Intelligence (AAAI), 2000, pp. 90–97.

[26] F. C. R. Spieksma, *On the Approximability of an Interval Scheduling Problem*, Journal of Scheduling, 2 (1999), pp. 215–227.

[27] J. M. van den Akker, C. A. J. Hurkens, and M. W. P. Savelsbergh, *Time-indexed formulations for single-machine scheduling problems: Column generation*, INFORMS Journal on Computing, 12 (2000), pp. 111–124.

[28] S. van Hoesel and R. Müller, *Optimization in electronic markets: Examples in combinatorial auctions*, Netnomics, 3 (2001), pp. 23–33.

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

[30] E. Wolfstetter, *Topics in microeconomics: industrial organization, auctions and incentives*, Cambridge University Press, Cambridge, 1999.

[31] P. R. Wurman and M. P. Wellman, *AkBA: A Progressive, Anonymous-Price Combinatorial Auction*, in Second ACM Conference on Electronic Commerce, 2000, pp. 21–29.