Using A Lagrangian Heuristic For A Combinatorial Auction Problem

Y. Guo[1], A. Lim[2], B. Rodrigues[3], J. Tang[2]
[1]School of Computing, National University of Singapore, 3
Science Drive 2, Singapore
[2]Department of IEEM, Hong Kong University of Science and
Technology, Clear Water Bay, Hong Kong
[3]Lee Kong Chian School of Business, Singapore Management University, Singapore

Abstract

In this paper, a combinatorial auction problem is modeled as a NP-complete set packing problem and a Lagrangian relaxation based heuristic algorithm is proposed. Extensive experiments are conducted using benchmark CATS test sets and more complex test sets. The algorithm provides optimal solutions for most test sets and is always 1% from the optimal solutions for all CATS test sets. Comparisons with CPLEX 8.0 are also provided, which show that the algorithm provides good solutions.

1 Introduction

The combinatorial auction problem we study can be described as follows. A supplier has a total of m jobs to finish and an auction is conducted to distribute jobs among a set of bidders. Bidders propose n bids, and each bid b_i ($1 \leqslant i \leqslant n$) covers a number, m_i ($1 \leqslant m_i \leqslant m$) of distinct jobs. If bid b_i is selected by the supplier, a profit w_i ($1 \leqslant i \leqslant n$) is resulted for the supplier, but any other bid that contained some same job as b_i cannot be selected. The goal is to maximize the total profit of the supplier without violating the constraint that each job can be contained in at most one selected bid.

DeVries and Vohra provided an excellent survey on combinatorial auctions [3]. Recently, the auction problem has been modeled as a set packing problem (SPP), a well-known NP-complete problem [9], [4], [1], [8], [5]. There have been many solution approaches suggested for this problem, including both exact and non-exact heuristic methods. Exact algorithms including a branch and bound search [4], iterative deepening A* search [8] and the direct application of available CPLEX IP solver [1] have been developed and applied to the SPP problem. A relatively common test set is CATS [7]. Experiments using CATS showed the CPLEX 6.5 solver to be a good approach among the exact methods [1]. Sandholm recently proposed a new branch-and-bound algorithm called CABOB, which when applied to the CATS test set with different distributions, provided generally results faster than CPLEX 7.0 [9]. CABOB and the CPLEX integer programming solver are the current leading exact methods for SPP. In order to cater for test sets with large scale or more difficult distributions, non-exact methods are often preferable. Non-exact algorithms, using iterative greedy heuristic [6], and stochastic local search [5] can be found in recent literature.

In this work, we develop a heuristic method based on the Lagrangian relaxation with subgradient optimization for the combinatorial problem described above. Variations of this technique have been applied to a closely related set covering problem [2], but to the best of our knowledge, no other work has used Lagrangian relaxation for the SPP. Experimental results show that the new approach compares well with other methods as a non-exact algorithm.

Recent research on SPP reveals that CATS generated test sets appear to be easy for exact methods like CPLEX [9]. Also from our experimental observations, CATS test sets are proved to be relatively easy for CPLEX because of the existence of dominating bids – bids that require few jobs but provide high profit. These dominating bids, however, are not realistic in real world situations. As a result, in addition to the 8 different distributions from CATS test set, we also designed our own test sets that features reasonable criteria such as that the profit of bid is roughly proportional to sum of the price of individual jobs it covers. And experimental results showed that CPLEX 8.0 would fail to provide the optimal solutions in reasonable time spent as it does for CATS, even when the test size is not very large.

The paper is organized as follows: in the next section we will give the formal definition of SPP, and discuss our LAHA heuristic in detail. The elaborate experimental results are presented in section 3, where results of our method are compared with results from CPLEX 8.0. Concerns with the test set generation together with CATS test set will be discussed. The paper gives the conclusion in section 4.

2 Lagrangian relaxation based heuristic on SPP

2.1 The combinatorial auction problem

The integer programming model for the SPP can be written as follows:

$$\text{maximize} \sum_{i \in N} w_i x_i \tag{1}$$

Subject to:

$$\sum_{i \in N} a_{ij} x_i \leq 1,\ j \in M \tag{2}$$

$$x_i \in \{0,1\},\ i \in N \tag{3}$$

where $N = \{1, \ldots, n\}$, $M = \{1, \ldots, m\}$ and $[a_{ij}]$ is an $n \times m$ 0-1 matrix where a_{ij} equals 1 only if bid i covers job j. The first set of constraints ensures that each row is covered by at most one column and the second integrality constraints ensure that $x_i = 1$, iff bid i is in the solution. This problem is a well-known NP-complete set packing problem, for which no polynomial time exact algorithm exists unless P = NP.

2.2 Lagrangian relaxed heuristic

The Lagrangian relaxation used in our algorithm is to find a nice upper bound of the IP model presented in the previous section, and then we base our heuristic to expand from the best upper bounds we have found to get feasible solutions with expected high quality.

According to the original IP model for SPP, for any Lagrangian multiplier vector $u = (u_1, u_2, ..., u_m)$ such that $u_j \geq 0$ for all $j \in M$, we define the following problem it SPP(u):

$$\text{maximize} \sum_{i \in N} w_i x_i + \sum_{j \in M} [u_j (1 - \sum_{i \in N} a_{ij} x_i)] \tag{4}$$

Subject to:

$$x_i \in \{0,1\},\ i \in N \tag{5}$$

The constraint(2) has been relaxed into the dual problem. The next, we need to prove that the dual problem is a relaxed problem of the original one.

Proposition 1 Problem $SPP(u)$ is a relaxation of the original problem SPP for any non-negative input of the $Lagrangian\ multiplier\ u$.

The proposition is true by the following two facts:

(1). Every feasible solution of SPP is a feasible solution of SPP(u) for arbitrary u.

(2). The solution value of SPP(u) is greater than or equal to the solution value of SPP for non-negative u.

From proposition 1, any solution x of SPP is also a solution for SPP(u), and the result of SPP(u) is an upper bound for SPP regardless of u. But proposition 1 alone is not enough to derive an upper bound for SPP, because the lagrangian multiplier u is not a constant. However, we know that given any u the optimal solution of SPP(u) can be found in linear time to mn by the following proposition and thus, it is an upper bound for SPP.

Proposition 2 When the Lagrangian multiplier u is fixed, the optimal solution of SPP(u) can be found in O(mn) time.

Proof According to the definition of $SPP(u)$ above, we have

$$\begin{aligned} SPP(u) &= \max\{\sum_{i \in N} w_i x_i + \sum_{j \in M} [u_j (1 - \sum_{i \in N} a_{ij} x_i)]\} \\ &= \max\{\sum_{i \in N} w_i x_i + \sum_{j \in M} u_j - \sum_{j \in M} (u_j \sum_{i \in N} a_{ij} x_i)\} \\ &= \max\{\sum_{i \in N} w_i x_i + \sum_{j \in M} u_j - \sum_{i \in N} \sum_{j \in M} u_j a_{ij} x_i\} \\ &= \max\{\sum_{j \in M} u_j + \sum_{i \in N} x_i (w_i - \sum_{j \in M} a_{ij} u_j)\} \end{aligned}$$

From the equation above, for a fixed u, $\sum_{j \in M} u_j$ is a constant, and $w_i - \sum_{j \in M} a_{ij} u_j$ is deterministic for any $i \in N$. So the optimal solution can be constructed by examining $w_i - \sum_{j \in M} a_{ij} u_j$ and let x_i be 1 if $w_i - \sum_{j \in M} a_{ij} u_j \geq 0$ or 0 otherwise. This algorithm runs in $O(mn)$.

From proposition 2, it is clear that by choosing a nice lagrangian multiplier, we can find a tight upper bound for SPP easily, and we will refine the solution found by algorithm described in proposition 2 to make it feasible for SPP. Next, we will look into how to find a suitable Lagrangian multiplier using subgradient optimization.

2.3 Subgradient optimization for the Lagrangian multiplier

The quality, or rather tightness of the upper bound found as in proposition 1, largely depends on the choice of Lagrangian multiplier. In our approach, we start with an initial vector of multipliers and then iteratively update the multiplier vector by an application of the subgradient optimization technique. We use u^j to denote the multiplier vector at iteration j, with the initial u^0 to be as follows.

$$u_j^0 = \frac{\sum_i^{a_{ij}=1} \frac{c_i}{\sum_k^{a_{kj}=1}}}{\sum_p^{a_{pj}=1}} \quad \forall j \in M \tag{6}$$

This actually takes into account for any job $j \in M$ the profit of a bid that covers job j divided by the number of jobs covered by that bid, and then average this variable to be u_j^0.

In order to discuss how to update u^k to get u^{k+1}, we need to introduce a new vector g, the subgradient vector, and use $u^{k+1} = u^k + g \times F$ with F a subgradient constant to obtain

u^{k+1}. A widely used scheme to decide the subgradient vector g, let $g_j(u) = 1 - \sum_i^{a_{ij}=1} x_i(u)$ for all $j \in M$. $x_i(u)$ is the solution of $SPP(u)$ given u found by using algorithm described in proposition 1.

In iteration k, the Lagrangian multiplier u_k is derived. And X_k, the optimal solution for SPP(u_k) can be obtained. This solution is probably infeasible for SPP so we need to adjust X_k to ensure the feasibility for the original set packing problem. We have used 2 methods to make a solution X_k feasible. The first one is a random heuristic, in which we first sort the bids selected in X_k in decreasing order of their reduced profit c_i, where $c_i = w_i - \sum_j^{a_{ij}=1} u_j^k$, $\forall i \in N$, which is also used in [2]. Then we will consider the bids one by one. Each bid has a probability of 0.9 to be selected, and a bid is only allowed to be selected if it does not contain any common job as the previously selected bids. This random heuristic will run 200 times for each X_k and thus 200 feasible solutions are obtained. In addition, another deterministic method is also used to obtain feasible solutions from X_k. Let the set T be the set of jobs covered by more than one bid in X_k, i.e, $job_j \in T$ iff $\exists bid_i \in N\ \exists bid_p \in N,\ bid_i \in X_k, bid_p \in X_k, a_{ij} = 1$ and $a_{pj} = 1$. The deterministic algorithm first include all bids selected in X_k, then exclude one bid at a time with the smallest $\frac{c_i}{|T \bigcap bid_i|}$, until $T = \emptyset$. Then the solution is feasible with respect to SPP. And a single feasible solution is obtained from each X_k. And among all the solutions we find, the best 50 are kept for later refinement as we will explain in the next section. The framework of our heuristic is elaborated in Algorithm 1.

Algorithm 1 Subgradient optimization to improve Lagrangian multiplier

$u_j \leftarrow \frac{\sum_i^{a_{ij}=1} \frac{c_i}{\sum_k^{a_{kj}=1}}}{\sum_p^{a_{pj}=1}} \quad \forall j \in M$
$F \leftarrow 1$
$iteration \leftarrow 0$
while $F > 0.01$ **do**
  $iteration \leftarrow iteration + 1$
  Calculate the solution X for u as in proposition 2
  Run the random heuristic 200 times to get feasible solution from X for SPP
  Run the deterministic algorithm to obtain one feasible solution from X
  Update the best 50 solutions
  $g_j \leftarrow 1 - \sum_i^{a_{ij}=1} X_i\ \forall j \in M$
  $u \leftarrow u + g \times F$
  if For $N1$ iterations, the 50 best solutions do not improve **then**
    F=F/2
  end if
end while

2.4 Refinement of solutions

After the 50 best solutions are found as described in the previous section, a greedy local search is applied to explore the vicinity of those solutions in the hope to improve the solution quality. Our experimental result, shows the greedy refinement is efficient to increase the solution quality by up to 3%. The refinement algorithm is described in Algorithm 2.

Algorithm 2 Greedy refinement of solution quality

$S \leftarrow set\ of\ bids\ appeared\ in\ any\ of\ the\ 50\ solutions$
while $|S| < 1500\ and\ S \leq n$ **do**
  Select a bid b_i with maximum profit among all bids not in S
  $S \leftarrow S \bigcup b_i$
end while
for $Each\ of\ the\ 50\ solutions\ t_i,\ i \leq 50$ **do**
  while $For\ some\ bid\ b_j \in S,\ that\ b_j \notin t_i\ and\ w_j > \sum_{p=1}^{n} w_p\ for\ b_p \in t_i\ and\ \exists k \in M,\ a_{pk} = a_{jk} = 1$ **do**
    $t_i = t_i - b_p\ for\ b_p \in t_i\ and\ \exists k \in M,\ a_{pk} = a_{jk} = 1$
    $t_i = t_i + b_j$
  end while
end for
Output the best among the 50 solutions

The set of bids S is the candidate set that we choose from to add to the 50 solutions. The constraint of maximum 1500 bids in S is to ensure the time efficiency of the refinement. The greedy method tries to add in new bids from the candidate set that after removing all conflicting bids with the new ones, will result in the increase of the total profit. The outer while loop is terminable since the profit of each t_i is monotonically increasing.

3 Experimental results

In this section we will present the experimental results of our LAHA heuristic and comparison with the CPLEX 8.0 integer programming solver. According to [9], the current leading exact algorithm, the branch and bound method CABOB outperforms CPLEX 7.0 moderately for most test sets they used from CATS. However, for 7 out of 9 distributions they used, both CPLEX 7.0 and CABOB generated optimal results within 10 seconds for the worst test case of each distribution. For another distributions that CABOB runs in 20 seconds for the worst test set while CPLEX 7.0 only needs less than 10 seconds. Only for 1 out of the 9 distributions, namely the components distribution, CABOB produces the optimal solution within 1 second and CPLEX 7.0 suddenly requires more than 800 seconds according to [9]. As a result, we believe CPLEX solver is still a compet-

itive tool to solve the SPP on CATS test sets. We also base our comparison with CPLEX 8.0, which is in general 40% faster than CPLEX 7.0 for integer programming problems according to Ilog company.

All experiments are done on a 2.8GHz Pentium(R) 4 machine with 1GB Memory. LAHA was implemented in ANSI C++ and compiled with GNU GCC 3.2 compiler. In order to conduct a fair comparison, parameters of CPLEX were tuned for SPP before the experiment. Because of the various natures and characteristics of our benchmarks, there is not a general parameter setting which works best for all kinds of instances. After careful considerations, we decided to set the parameters for CPLEX 8.0 as follows: 1. set mip strategy to emphasize feasibility; 2. set mip clique cut generation strategy to 2 (aggressive); 3. explore the "up" branch first in the enumeration tree. Default values were used for all the other parameters. We find that the above setting works especially well for the difficult benchmarks.

3.1 Comparison using CATS test set

We have used a total of 8 different distributions from CATS, namely the **exponential, random, uniform, binomial, decay, scheduling, matching** and **paths** distributions. Test cases of the first 5 distributions are exactly the same as used in [1], and we generated the later 3 extra distributions directly from CATS. We will follow the name convention in describing the results. For example, $exp-p-q$ is the test set of exponential distribution with p jobs and q bids.

The results are presented in Table 1.$\mu_{Density}$ is the average density of the 10 test set in each group. The density of a test set is defined as $\frac{\sum_{i\in N}\sum_{j\in M} a_{ij}}{n\times m}$ which is an indicator as how many jobs one bid covers. μ_{Cplex} and μ_{LAHA} is the average result CPLEX 8.0 and LAHA obtained for each category. t_{CPLEX} and t_{LAHA} is the average time spent in seconds. δ is the difference between LAHA's result and CPLEX's optimal result.

From this table we see that for all CATS test sets LAHA provides high quality results. For exponential, random, binomial and scheduling distributions LAHA get all optimal results while for the other 4 distributions the results are all within 1% of optimal. With respect to time efficiency, for most test sets except the uniform, random and binomial distributions, CPLEX is very efficient as it provides optimal results within several seconds. On the other hand, LAHA's time spent is longer than CPLEX for those test sets but the efficiency is still very high, providing quality results in tens of seconds. However, LAHA's time spent from 0.8 to 54 seconds, has a smaller variation than CPLEX, which runs in 0.2 to 230 seconds. This, to our belief is due to the fact that LAHA is a heuristic with relative stable computational complexity, and CPLEX's branch and cut algorithm would largely depends on the nature of test sets.

3.2 Comparison using PBP test set

In this section, another test set different from CATS is used to compare the performance of CPLEX 8.0 and LAHA. Some of the CATS generated test cases appeared to be easy. After careful investigations, we have found out because CATS's generation mechanism may lead to lots of dominating bids in the test cases. The problems' scale can usually be decreased dramatically after the dominance has been removed. And usually the LP relaxation of those CATS problems is very close, if not equal, to its original IP, thus it made such problems trivial for any LP based Branch and Bound solvers. However, for the real world combinatorial auction problems, the price for a bid is usually related to the quantity and quality of the jobs which it covers. While the real value for a job may fluctuate according to the market changes, it is usually proportional to the number of bids which cover it. We now propose a new methodology to produce a Proportional Bid Price(PBP) SPP test set based on this observation. The test cases turn out be hard for CPLEX to solve with only hundreds of rows and columns. The PBP test sets are generated as described in Algorithm 3.

Algorithm 3 Price Proportional Bid Test Set Generation

Specify n,m and probability density for coefficient matrix a
Step 1: Generate coefficient matrix a
for all bid i from 1 to n **do**
  for all job j from 1 to m **do**
    $a_{ij} \leftarrow 1$ with probability density specified
  end for
end for
Step 2: Generate price for the jobs
for all job j from 1 to m **do**
  price of $job_j \leftarrow$ # of bids which covers job_j from $a\times f$, where f is a random in the range (0.9, 1.1)
end for
Step 3: Generate the price for the bids
for all bid i from 1 to n **do**
  price of $bid_i \leftarrow$ summation of the price of the jobs it covers $\times$ f, where f is random in (0.9, 1.1)
end for
Output the coefficient matrix a and the prices for the bids.

We have generated test cases from PBP of different sizes. Sine CPLEX 8.0 could not give the optimal results for many test cases within reasonable time spent, we have set the time limit of CPLEX for 1800 seconds. The results are presented in Table 2. δ_1 is the ratio of LAHA's result over CPLEX's, and D is the density.

Among all the 15 test cases we generated as in Table 2, LAHA obtained 14 results as good as CPLEX 8.0, and 4 results outperformed CPLEX by 2% to 11%. And one is 1% worse than CPLEX result. From the table, we see that LAHA at least obtained 8 optimal solutions for

Test set	# instance	$\mu_{Density}$	μ_{CPLEX}	t_{CPLEX}	μ_{LAHA}	t_{LAHA}	δ
EXP-30-3000	10	0.09	44723.8	0.27	44723.8	0.81	0
RND-400-2000	10	0.50	16143.8	5.14	16143.8	14.62	0
UNI-100-500	10	0.03	129050.0	31.31	128254	2.3954	0.6%
BIN-150-1500	10	0.20	94792.7	234.00	94792.7	7.91	0
DEC-200-10000	10	0.02	196266.0	34.81	194957.0	54.01	0.7%
SCH-400-2200	10	0.02	54.256	0.27	54.256	25.91	0
MAT-600-2000	10	0.01	997.419	0.26	997.218	12.63	0.02%
PAT-600-2000	10	0.01	61.898	0.46	61.852	19.28	0.07%

Table 1. Experimental results on CATS test set

Test set	D	$LAHA$	t_{LAHA}	$CPLEX$	t_{CPLEX}	δ_1
100-200	0.05	901.42$\star$	0.984	901.42	5.797	1.00
100-200	0.10	1428.17$\star$	0.844	1428.17	11.407	1.00
100-200	0.15	1818.26$\star$	0.625	1818.26	4.75	1.00
200-200	0.03	946.62$\star$	1.547	946.62	23.594	1.00
200-200	0.05	1296.03$\star$	1.281	1296.03	97.360	1.00
200-200	0.10	1916.20$\star$	0.921	1916.2	11.172	1.00
200-1500	0.10	18118.9	7.953	17486.4	tle	1.04
200-1500	0.15	21937.5$\star$	6.781	21937.5	972.703	1.00
200-1500	0.20	25583.9$\star$	8.453	25583.9	1060.92	1.00
500-2000	0.03	17598.5	21.41	17312.2	tle	1.02
500-2000	0.10	33883.2	12.75	32284.8	tle	1.05
500-2000	0.15	47070	15.91	42329.9	tle	1.11
500-5000	0.05	58210.2	27.31	58741.2	tle	0.99
500-5000	0.15	121007.0	42.67	121007.0	tle	1.00
500-5000	0.20	148706.0	38.17	148706.0	tle	1.00

[1] $\star$ means optimal solution is found

[2] tle means CPLEX cannot obtain the optimal result within 1800 seconds, and the best result at that time is reported.

Table 2. Experimental results on PBP test set

which CPLEX has successfully acquired within 1800 seconds. Notably, LAHA's time efficiency is much better than CPLEX, always producing the quality result within 15%, and often less than 1% the time required by CPLEX 8.0. The experiments on PBP test sets showed our heuristic is an compelling approach to SPP.

4 Conclusion

In this paper, the combinatorial auction problem is modeled as a NP-complete set packing problem. And various approaches to this problem in literature are discussed. We proposed a new Lagrangian heuristic LAHA to solve this problem. This non-exact approach showed reasonably good performance when compared with the currently leading exact CPLEX 8.0 solver on various CATS test sets, by providing optimal results for half the test cases and 1% form the optimal for the rest. When applied on the Price Proportional Bid (PBP) test sets, LAHA provides equally good or better results than CPLEX with always less than 20% and mostly less than 1% time required by CPLEX.

References

[1] A. Andersson, M. Tenhunen, and F. Ygge. Integer programming for combinatorial auction winner determination. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, pages 39–46, 2000.

[2] A. Caprara, M. Fischetti, and P. Toth. A heuristic method for the set covering problem. *Lecture Notes in Computer Science 1084, Springer, 72–84.*, 1995.

[3] Sven de Vries and Rakesh V. Vohra. Combinatorial auctions: A survey. *INFORMS Journal on Computing*, (3):284–309, 2003.

[4] Y. Fujishima, K. Leyton-Brown, and Y. Shoham. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *Sixteenth International Joint Conference on Artificial Intelligence*, pages 548–553, 1999.

[5] Holger H. Hoos and Craig Boutilier. Solving combinatorial auctions using stochastic local search. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, pages 22–29, 2000.

[6] Hoong Chuin Lau and Yam Guan Goh. An intelligent brokering system to support multi-agent web-based 4th-party logistics. In *Proceedings of the Fourteenth International Conference on Tools with Artificial Intelligence*, pages 10–11, 2002.

[7] Kevin Leyton-Brown, Mark Pearson, and Yoav Shoham. Towards a universal test suite for combinatorial auction algorithms. In *ACM Conference on Electronic Commerce*, pages 66–76, 2000.

[8] Tuomas Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135(1-2):1–54, 2002.

[9] Tuomas Sandholm, Subhash Suri, Andrew Gilpin, and David Levine. CABOB: A fast optimal algorithm for combinatorial auctions. In *IJCAI*, pages 1102–1108, 2001.