Heuristics for a Brokering Set Packing Problem

Y. Guo[1], A. Lim[2], B. Rodrigues[3], Y. Zhu[2]

[1]Dept of Computer Science, National University of Singapore

Science Drive 2, Singapore 117543

guoyunso@comp.nus.edu.sg

[2]Dept of IEEM, Hong Kong University of Science Technology

Clear Water Bay, Kowloon,Hong Kong

{iealim,zhuyi}@ust.hk

[3]School of Business, Singapore Management University

469 Bukit Timah Road, Singapore 259756

br@smu.edu.sg

Abstract

In this paper we formulate the combinatorial auction brokering problem as a Set Packing Problem (SPP) and apply a simulated annealing heuristic to solve SPP. Experimental results are compared with a recent heuristic algorithm and also with the CPLEX Integer Programming solver where we found that the simulated annealing approach outperforms the previous heuristic method and CPLEX obtaining results within smaller timescales.

1 Introduction

In [5], a model was proposed for modeling a brokering problem as a set packing problem (SPP). This problem was motivated by the need to provide an intelligent agent-based framework that supports fourth-party logistics operations. Combinatorial optimization problems on bidding auction have been extensively studied in the literature [1][2]. Recently, combinatorial auction theory has become a subject of interest. Vries and Vohra provide an excellent survey for such problems, including the SPP and two other relevant problems, the set partition and covering problems[7]. While there are some previous papers on the approaches to set partition and covering problems, set packing problem is relatively less analyzed [3][4][6].

In this work, we study the broker model which can be described as follows: Assume there are n bids and m jobs and each bid can cover a number of jobs resulting in a profit to the supplier, w_j $(j \in 1, ..., n)$ is the profit if bid j is selected, and $[a_{ij}]_{m \times n}$ is a m-row, n-column 0-1 matrix, where $a_{ij} = 1$ if job i is included in bid j. Further, the decision variables, $x_j = 1$ if the supplier selects bid j, and 0 otherwise. An integer programming (IP) model for the brokering problem as a SPP problem is then:

$$\text{Maximize} \sum_{j \in N} w_j x_j \tag{1}$$

Subject to:

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

$$x_j \in \{0, 1\}, \ j \in N \tag{3}$$

where $N = \{1, \ldots, n\}$, $M = \{1, \ldots, m\}$.The first set of constraints ensure that each row is covered by at most one column and the second integrality constraints ensure that $x_j = 1$, iff column j of the matrix is in the solution.

In [5], the SPP is directly related to bid allocation since rows represent jobs submitted and columns represent bids for a subset of rows and the problem is to find a set fo bids with the maximum weight which cover a given set of jobs at most once. The weights were derived by the three factors of pricing, preference and fairness and is indicative of how good any bid is. Although the SPP is well known, we have to the best of our knowledge not found any application of heuristic to this problem, except in [5] where an Iterative Greedy (IG) approach was proposed for the problem. The IG method is applied to problems of up to 200 jobs and 100 bids. In this paper, we develop a heuristic method using an simulated annealing heuristic embedded with a greedy search which provides a guide to the main algorithm. From extensive experiments performed, we found that this approach gave better results with less time spent compared with the IG approach. And when applied to large test cases with up to 1500 jobs and 1500 bids, the new method gave good results compared with those obtained by CPLEX and in much less time.

The paper is organized as follows: in the next section we propose a new simulated annealing heuristic embedded with greedy local search to solve SPP and also discuss the solution techniques in detail. The elaborate experimental results are presented in section 3, where the results of our method are compared with CPLEX solver and the previous heuristic method. The paper gives the conclusion in section 4.

2 Simulated Annealing with Greedy Search

Simulated Annealing (SA) is a meta-heuristic that differs from the traditional hill-climbing method. It accepts local moves which may decrease the current objective value with a certain probability. It comprises of two major components - a local search and temperature cooling schedule. We developed a hybrid meta-heuristic which basically uses an SA approach and with a greedy search for selecting local moves.

2.1 The SA Framework

Our heuristic algorithm is illustrated in Algorithm 1, where the $m \times n$ matrix a denotes a job-bid 0-1 matrix where $a_{ij} = 1$ if bid j includes job i ($1 \leq i \leq m$, $1 \leq j \leq n$). Before the program embarks on the heuristic search, we preprocess input data to eliminate redundant calculations possible in the heuristic. The preprocessing would first calculate a 0-1 matrix $C = [c_{ij}]_{n \times n}$, where $c_{ij} = 1$ if bid i and bid j both contain job k ($1 \leq i \leq n$, $1 \leq j \leq n$, $1 \leq k \leq m$), and bid i and bid j are called "conflict" each other. If there exists any bid i ($1 \leq i \leq n$) such that $c_{ij} = 0$ for any j ($1 \leq j \leq n$), we include the bid i in the final solution and do not consider such bids in the heuristic search.

Here, we combine the greedy local search with the SA search as the greedy method by itself would easily go into local optima but if it is used to guide the SA search, the results turn out much better than the IG method in [5] or if we use SA alone. The extent to which the greedy

Algorithm 1 Simulated Annealing Framework

$S \leftarrow \{\}$; $best\_value \leftarrow 0$; $Temperature \leftarrow T_{max}$; $Iter \leftarrow 0$
$preprocess()$
while $Iter < Max\_Iter$ and $Temperature > T\_Terminate$ **do**
  with probability p_1
    $S \leftarrow SA\_Localsearch(s)$;
    Cool($Temperature$)
  with probability $(1 - p_1)$
   $S \leftarrow Greedy\_Localsearch(s)$;
  if $value(S) > best\_value$ **then**
    $best\_value \leftarrow value(S)$;
  end if
  $iter \leftarrow iter + 1$
end while

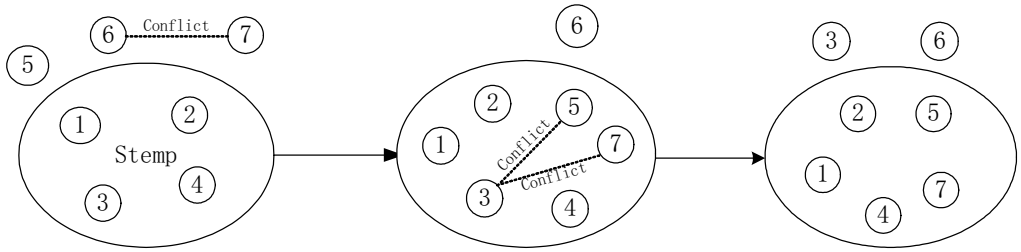

Figure 1: SA_exchange_2 Example

local search is embedded with the SA search is an important factor, and in our implementation, this is expressed by the probability p_1.

2.2 SA Local Search and Cooling Schedule

Unlike hill climbing, SA local search can accept moves that will lower the current objective value in order to escape from possible local optima. The SA local search algorithm is sketched in Algorithm 2, where C_4 is a constant.

We have made use of 2 types of neighborhood moves: $SA\_exchange\_1$ and $SA\_exchange\_2$. These would select 1 or 2 bids at one time respectively to add to the current solution S, and then remove any bids in S that conflict with the newly added bids. For example, the algorithm for $SA\_exchange\_2()$ is illustrated by Algorithm 3. Also refer to Fig 1. In this example, we have 7 bids in all with 4 already selected in *Stemp*. $SA\_exchage\_2$ would select 2 non-conflicting bids 5, 7 from outside *Stemp* to add to it, and remove bid 3 from *Stemp* because it conflicts with the newly added bids. For a cooling scheme, we set the initial temperature T_{max} to have a large value with a cooling schedule $Temperature = C_5 \times Temperature$ where C_5 is a constant a little smaller than 1.

2.3 Greedy Local Search

The greedy local search is used with the SA local search to improve the performance. In a greedy local search, the greedy value determination is directly related to the performance of

Algorithm 2 SA local search

$Stemp \leftarrow S$
with probability p_2
SA_exchange_1($Stemp$)
with probability $1 - p_2$
SA_exchange_2($Stemp$)
$\delta = value(Stemp) - value(S)$
if $\delta > 0$ **then**
return $Stemp$
else {from S to $Stemp$ is a downhill move}
$T \leftarrow C_4 \times Temperature$
$p = e^{-\delta/T}$
with probability p
$S \leftarrow Stemp$
return S
end if

Algorithm 3 SA_exchange_2($Stemp$)

if At least 2 non-conflict bids are not in $Stemp$ **then**
Randomly select bid i and bid j where $i, j \notin Stemp$ and $c_{ij} = 0$
Remove bid k from $Stemp$ if $c_{ik} = 1$ or $c_{jk} = 1$
$Stemp \leftarrow Stemp \cup \{i, j\}$
end if
return $Stemp$

the search. In SPP, a natural approach is to use the profit of a bid as the greedy value. However, this approach ignores the nature of a bid in the way that more jobs a bid contains, the more likely the bid would conflict with other bids and constrain bids that the supplier can select which makes solutions inferior. In our greedy local search, we assign a penalty cost to each bid, and take the profit less the penalty cost as the greedy value.

2.3.1 Relative Penalty Cost

In order to derive the relative penalty cost, we first introduce an absolute penalty cost $|penalty|$ given by:

$$|penalty|_i = \sum_{j=1}^{n} C_1 * profit_j * c_{ij},\ 1 \leqslant i \leqslant n.$$

$[c_{ij}]_{n\times n}$ is a n-row, n-column 0-1 matrix where $c_{ij} = 1$ iff $\exists k \in$ M $a_{ik} = 1$ and $a_{jk} = 1$.

When bid i conflicts with a high-profit bid j, if we select bid i and not bid j, the outcome may be detrimental; thus, we assign a penalty cost to each bid according to the profits accrued from all other conflicting bids as above. Here, C_1 is a constant.

We can now define a relative penalty:

$$penalty_i = \sum_{j=1}^{n} (C_2 * profit_j - C_3 * |penalty|_j) * c_{ij},\ 1 \leqslant i \leqslant n.$$

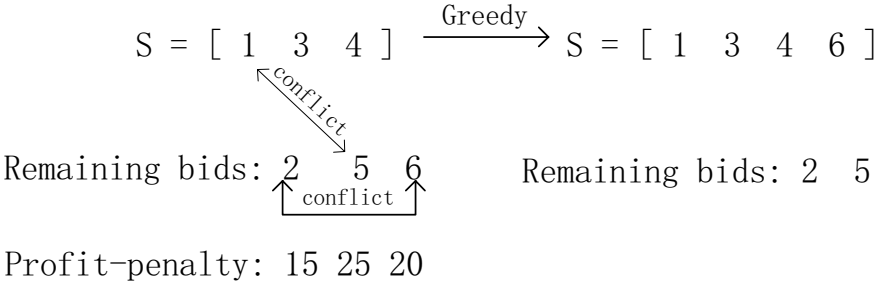

Figure 2: Greedy Search Example

The main advantage that a relative penalty has over an absolute penalty is that, when bid j has a large absolute penalty and we are unlikely to pick it in the greedy process, we need not incur the absolute penalties of other bids that conflict with bid j since we subtract $C_3 * |penalty|_j$. Here, C_2 and C_3 are both constants.

2.3.2 Greedy Selection

The greedy local search chooses among the bids that do not conflict with any of the bids already selected in S one with the largest $(profit - penalty)$ value. As in Fig. 2, suppose we have 6 bids and we have selected current bids $S = \{1, 3, 4\}$ where there are conflicts between bid 1,5 and bid 2,6. After the greedy local search is applied, the resulting set is $S = \{1, 3, 4, 6\}$.

3 Experimental results

In this section, we present experimental results and compare these with the IG method in [5] and Ilog CPLEX 7.0.1 using the SPP IP model described in this paper. We generated the test cases following the mechanisms described in [5] with various sizes. These involve restricting the number of jobs a bidder can bid and the minimum and maximum profit each job can provide, etc. In [5] the maximum size of test case is $m = 200$ and $n = 100$ and in this work, we use values up to $m = 1500$ and $n = 1500$. For comparison with our SA with greedy search (SAG), we tested 4 variations of the IG method in [5] and from our experimental results all 4 methods do not significantly differ from each other in results. For comparisons, we use the second variation of the IG method, a deterministic divergent search and deterministic partial cover described in [5]. All instances on SAG and IG are run on a machine with PIII-800 dual CPU and 1GB memory; all instances on CPLEX are run on a local machine with PIV-1.7G CPU and 384M memory.

3.1 Optimality Comparisons of SAG and IG with CPLEX

We randomly generated 10 small test cases following the generation mechanism introduced in [5] for which CPLEX can provide optimal results in reasonable time and were able to compare the performance of SAG and IG for small instances. The results are given in Table 1. The time spent on these 10 instances is less than 10 seconds for both SAG and IG methods; and less than 30 seconds for CPLEX.

M	N	SAG	IG	δ_{IG}[1]	CPLEX
100	100	19495.02†[2]	17400.77	10.74%	19495.02
100	100	20543.42†	20435.92	0.52%	20543.42
100	100	18556.59†	16993.77	8.42%	18556.59
100	100	20690.86†	19590.13	5.32%	20690.86
100	100	18485.96†	18485.96†	0	18485.96
200	200	26284.21†	23920.52	8.99%	26284.21
200	200	28473.75†	26142.23	8.19%	28473.75
200	200	27894.38†	27894.38†	0	27894.38
200	200	30748.64†	30748.64†	0	30748.64
200	200	27558.78†	25636.82	6.97%	27558.78

[1] δ_{IG} measures the difference between IG and optimal solution

[2] † means optimal solution is obtained

Table 1: Optimality Comparison for Small Instances

M	N	# instance	SAG best	IG best	μ_{SAG}.	t_1	μ_{IG}.	t_2	δ[1]
1000	500	100	94	1	74750.66	33.88	70295.46	31.73	6.34%
500	1000	100	91	8	65874.39	77.06	62732.89	143.62	5.01%
1000	1000	100	94	5	85205.58	88.70	80256.87	188.29	6.17%
1000	1500	100	100	0	84422.91	151.50	80404.98	286.84	5.00%
1500	1500	100	93	1	103426.68	192.93	98255.77	314.04	5.26%

[1] δ measures the percentage SAG result outperforms IG result

Table 2: Experimental Results on Large Instances

From Table 1, we see that SAG can find optimal solutions for the 10 small instances, while IG finds 3 out of 10. We can conclude that SAG works better than IG for small size instances.

3.2 Comparisons between SAG and IG Heuristics on Large Instances

In order to compare the performance of the SAG and IG heuristic in real world situations, we generated several sets of large test cases of the SPP and apply SAG and IG to solve. 500 test cases are grouped into 5 sets up to $m = 1500$ and $n = 1500$ as in Table 2, where μ_{SAG} and μ_{IG} are the arithmetic average of the 100 instances in each group for SAG and IG method respectively, and t_1 and t_2 are arithmetic average times in seconds of 100 instances in each group for the SAG and IG methods.

From Table 2, we see that the SAG method always gives a 5 to 6 percent improvement in results. The number of instances of each 100-instance group for our SAG method wins over IG is more than 90, which means SAG consistently outperforms the previous IG method. In addition, the time spent of SAG is always about 1/2 that of IG under same conditions excluding the first set of test cases which has a relatively small size and both SAG and IG run in about 30 seconds where SAG results are more than 6% better. In addition, the 500 instances are available at http://logistics.ust.hk/˜ zhuyi/instance.zip, to serve as our benchmark for the SPP.

M	N	SAG	t_1	$CPLEX$	δ_{CPLEX}[1]	IG	t_2	δ_{IG}[1]
1000	1000	87138.76	87.64	77977.06	10.51%	80712.11	143.29	7.37%
1000	1000	82914.83	88.89	81398.69	1.83%	77132.22	130.24	6.97%
1000	1000	85536.72	86.63	77995.93	8.81%	81958.07	134.54	4.18%
1000	1000	83254.88	88.82	74232.00	10.84%	74157.05	141.06	10.93%
1000	1000	87177.65	89.92	80210.71	7.99%	81097.49	141.13	6.97%
1500	1500	108008.96	193.11	98482.81	8.82%	99975.09	306.34	7.44%
1500	1500	100897.01	190.46	99377.82	1.51%	93638.39	322.40	7.19%
1500	1500	104749.69	192.50	94153.32	10.12%	100033.68	329.04	4.50%
1500	1500	101049.25	193.05	91548.39	9.40%	94955.79	321.58	5.99%
1500	1500	100658.78	192.08	97390.90	3.25%	96419.52	318.41	4.21%

[1] δ_{IG} and δ_{CPLEX} measure the result difference of IG and CPLEX from SAG

Table 3: Comparision among SAG, Cplex and IG on Large Instances

3.3 Comparisons between SAG and CPLEX on Large Instances

We randomly selected 10 instance from section 3.2 with size $m = 1000$, $n = 1000$ and $m = 1500$, $n = 1500$. We give CPLEX 3600 seconds time limit to run each instance (Hence the CPLEX solutions may not optimal) and compare the results. The statistics are found in Table 3.

From Table 3 we see that the optimal solution cannot be found by CPLEX when instance size is large. On the other hand, the SAG heuristic provides results which are reasonably better than CPLEX and the IG heuristic with higher time efficiency. Also, we note that IG does not provide better results than CPLEX always, whereas the SAG method does.

4 Conclusion

In this paper, we modelled the combinatorial auction brokering problem as a NP-complete set packing problem and discussed a meta-heuristic hybrid method to solve the problem. Although almost all approaches to the SPP have used IP techniques, we implemented a simulated annealing heuristic which provides good results with less time spent. When compared with the single other paper which uses heuristics for the SPP, our algorithm fairs better. Similarly, when compared with IP solutions obtained from CPLEX, our heuristics performs better. We also established our benchmark set for future research on SPP.

References

[1] M. Tenhunen A. Anderson and F. Ygge. Integer programming for combinatorial auction winner determination. In *Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS00)*, pages 39–46, 2000.

[2] M. Aourid and B. Kaminska. Neural networks for the set covering problem: An application to the test vector compaction. In *IEEE international Conference on Neural Networks Conference Proceedings*, volume 7, pages 4645–4649, 1994.

[3] E. Balas and A. Ho. Set covering algorithm using cutting planes, heuristics, and subgradient optimization: A computational study. In *Mathematical Programming*, volume 12, pages 37–60, 1980.

[4] E. Balas and M. W. Padberg. Set partitioning: A survey. In *SIAM Review*, volume 18, pages 710–760, 1976.

[5] 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.

[6] M. W. Padberg. On the facial structure of set packing polyhedra. In *Mathematical Programming*, volume 5, pages 199–215, 1973.

[7] Sven De Vries and Rakesh V. Vohra. Combinatorial auctions: A survey. In *INFORMS Journal on Computing*, volume 15, pages 284–309, 2003.