SUPPLEMENTARY FILE

Modeling and Solving the Supply Marketing Order Allocation Problem with Time Consistency and Bundle Discounts

Yupeng Zhou [a], Minghao Liu [b], Feifei Ma [c,d], Na Luo [a] and Minghao Yin [a,e,*]

[a]School of Computer Science and Information Technology, Northeast Normal University, Changchun, 130117, China;
[b]University of Chinese Academy of Sciences, Beijing, 100049, China;
[c]State Key laboratory of Computer Science, Institute of Software Chinese Academy of Sciences, Beijing, 100190, China
[d]Laboratory of Parallel Software and Computational Science, Institute of Software Chinese Academy of Sciences, Beijing, 100190, China
[e]Key Laboratory of Applied Statistics of MOE, Northeast Normal University, Changchun 130024, China

ARTICLE HISTORY
Compiled March 19, 2021

1. Problem Description

Before proceeding, some nomenclature and abbreviations are given in Table 1.

Table 1. Nomenclature and abbreviations

Abbreviation	Description
SMOAP	Supply Marketing Order Allocation Problem
SSOA	Supplier Selection and Order Allocation
CDA	Combinatorial Double Auction
SPP	Set Packing Problem
SMT	Satisfiability Modulo Theories
ILP	Integer Linear Programming
MILP	Mixed Integer Linear Programming
CP	Constraint Programming
LS	Local Search
TS	Tabu Search
GRASP	Greedy Randomized Adaptive Search Procedure

1.1. Example

The supply marketing order allocation problem can be treated as a constraint weighted bipartite graph matching problem, which is easier to understand. Purchasers' orders and suppliers' orders are naturally divided into two subgraphs. The vertices in each subgraph represent submitted orders and the edges connecting two vertices stand

* Corresponding author: Minghao Yin (ymh@nenu.edu.cn)

for feasible matching. The weight of each vertex is the order price, and the weight of each edge is the delivery cost. For suppliers, vertices with the same font color are in conflict, which is discussed above. So they cannot be allocated to the same demand order. Figure 1 shows an example of SMOAP. Suppose there are six purchasers $P1, P2, P3, P4, P5, P6$ and nine suppliers $S1, S2, S3, S4, S5, S6, S7, S8, S9$. In the graph, $S1, S6$ are conflicting, $S2, S3, S4$ are conflicting, and $S8, S9$ are conflicting. $S1$ bids for $(P4, P5)$, $S2$ bids for $(P1, P6)$, $S3$ bids for $(P1, P2, P3)$, $S4$ bids for $P3$, $S5$ bids for $P2$, $S6$ bids for $(P4, P5)$, $S7$ bids for $P5$, $S8$ bids for $P1$, and $S9$ bids for $(P2, P3)$. An allocation plan is $\{< P1, S2 >, < P1, S8 >, < P2, S3 >, < P2, S5 >, < P2, S9 >, < P3, S4 >, < P4, S1 >, < P5, S6 >, < P5, S7 >\}$ with an objective function value $f = (W1_{price} + W2_{price} + W3_{price} + W4_{price} + W5_{price}) - (W'1_{price} + W'2_{price} + W'3_{price} + W'4_{price} + W'5_{price} + W'6_{price} + W'7_{price} + W'8_{price} + W'9_{price}) - (E1_{delivery} + E2_{delivery} + E3_{delivery} + E4_{delivery} + E5_{delivery} + E6_{delivery} + E7_{delivery} + E8_{delivery} + E9_{delivery})$. Our aim is to provide an optimal plan that can maximize f, i.e., the profits of the platform. For those blue nodes, agriculturists are suppliers, while for green nodes, they are purchasers. However, these two trading backgrounds can be integrated into one model and solved together.

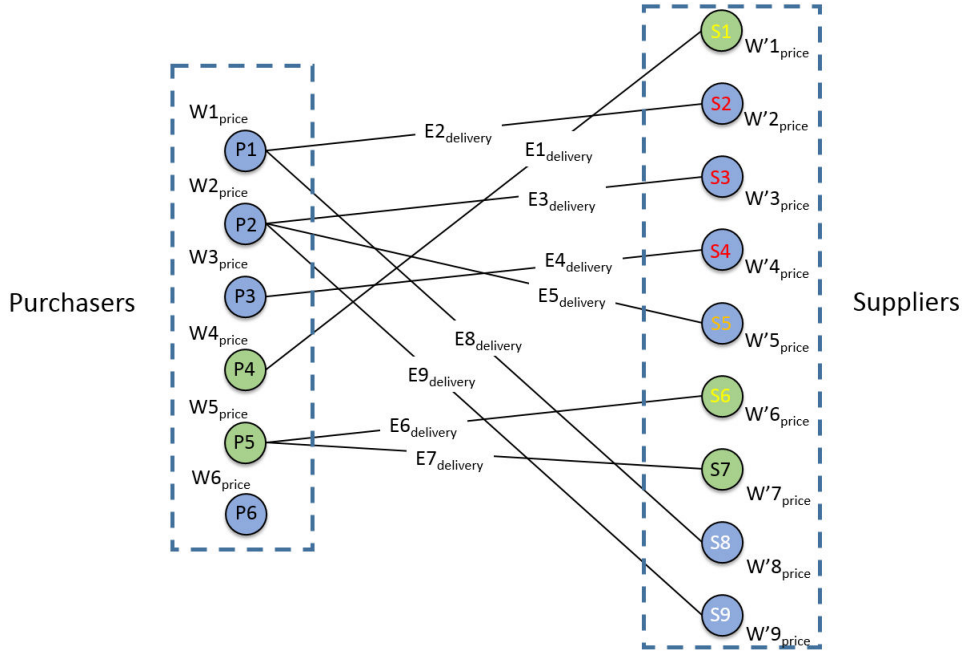

Figure 1. An example of SMOAP

1.2. NP-hardness of SMOAP

We now prove the NP-hardness of SMOAP. Different from the combinatorial auction problem which can be modeled as a set packing problem (SPP), SMOAP is proved via reduction from the exact cover problem.

Proposition 1. *The supply marketing order allocation problem is NP-hard.*

Proof. Consider the more simplified decision version of SMOAP, which asks if it is possible to allocate some supply orders from O^S to only one demanding order from O^P so that the constraints 1, 3 and 4 are satisfied. For constraints 2 and 5, we can assume that the purchaser always pays enough money, and the time segment of every supply order has an intersection with the demanding order. We can prove that this problem is NP-complete via reduction from the exact cover problem, which is one of Karp's 21 NP-complete problems. The exact cover problem is: Given a finite set X and a collection C of subsets of X, does C contains an exact cover for X, that is, a subcollection $C' \subseteq C$ such that every element of X occurs in exactly one member of C'? We can construct a simplified decision problem of SMOAP from the exact cover problem by the following steps:

(1) For the finite set X, create a demanding order o_i^P, whose demanding set of goods equals X.
(2) For the collection C, create a supply order o_j^S for each of its elements C_j, whose supply set of goods equals C_j.

Apparently, the above procedure is a polynomial-time reduction. Suppose the SMOAP has a solution $S = \{x_{i,j}\}$, then every C_j whose corresponding variable $x_{i,j}$ is **true** can form a subcollection C', which is a solution of the exact cover problem. Conversely, if the exact cover problem has a solution, so does the SMOAP, too.

Now that we have proved the NP-completeness of the simplified decision version of the SMOAP, it easily follows that the problem itself is NP-hard. □

2. The Local Search Algorithm

2.1. Solution representation and search space

Given a SMOAP model with $|S|$ suppliers, $S = \{S_1, S_2, ..., S_n\}$, and $|P|$ purchasers, $P = \{P_1, P_2, ..., P_m\}$, a feasible allocation can be denoted as a binary tuple $\{a_{ij} =< P_i, S_j >: P_i \in P, S_j \in S\}$ such that $a_{ij} = 1$ if supply order S_j is allocated to purchaser order P_i, and $a_{ij} = 0$ otherwise. Based on this, any candidate solution A is represented by k-dimensional vectors, i.e., $A = \{a_{ij} = 1 : \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} = k, 1 \leq i \leq m, 1 \leq j \leq n\}$. We further define the search space as $\Omega = \{a : a \in \{0,1\}^{m*n}\}$. It should be noted that k is unfixed according to the characteristics of the real scenario. That is, unreasonably submitted orders of both sides may exist in the data gathering module of the platform, thus causing less successful allocations (a smaller value of k). In this case, the lower bound of k is assigned 0. As each supply order is only allowed to bid for a unique demand order, the upper bound of the cardinality k is set to n.

2.2. Preprocessing Stage

In general, a lot of data will be included in one e-commerce platform, some of which are redundant to the computation module. As we focus on algorithmic aspects of this problem, the data preprocessing is needed to simplify the incoming inputs. Thus, only the following variables are left behind per instance: *total goods number*, *purchaser number*, *supplier number*, *orders*(*ID*, *price*, *commodity set*, *weight*, [$date_{start}$, $date_{end}$], *district code*). By this means, instances for the supply marketing allocation problem are simplified.

Table 2. Reduction rate of preprocessing

Name	Before reduction				After reduction				Reduction rate		
	suppliers	purchasers	variables	constraints	suppliers	purchasers	variables	constraints	suppliers	variables	constraints
SMOAP-1	186	8	1682	2804	113	8	1025	2074	39.25%	39.06%	26.03%
SMOAP-2	182	8	1646	2764	139	8	1259	2334	23.63%	23.51%	15.56%
SMOAP-3	201	8	1817	2970	126	8	1142	2220	37.31%	37.15%	25.25%
SMOAP-4	189	8	1709	2834	125	8	1133	2194	33.86%	33.70%	22.58%
SMOAP-5	248	10	2738	4416	137	10	1517	3084	44.76%	44.59%	30.16%
SMOAP-6	425	21	9371	16873	287	21	6335	13699	32.47%	32.40%	18.81%
SMOAP-7	432	21	9525	16740	284	21	6269	13336	34.26%	34.18%	20.33%
SMOAP-8	366	21	8073	14676	246	21	5433	11916	32.79%	32.70%	18.81%
SMOAP-9	382	21	8425	15338	285	21	6291	13107	25.39%	25.33%	14.55%
SMOAP-10	402	21	8865	16428	271	21	5983	13415	32.59%	32.51%	18.34%
SMOAP-11	1175	40	48215	89830	813	40	33373	74626	30.81%	30.78%	16.93%
SMOAP-12	1159	40	47559	88918	805	40	33045	74050	30.54%	30.52%	16.72%
SMOAP-13	1138	40	46698	88036	753	40	30913	71866	33.83%	33.80%	18.37%
SMOAP-14	1185	40	48625	91130	842	40	34562	76724	28.95%	28.92%	15.81%
SMOAP-15	1242	40	50962	97124	871	40	35751	81542	29.87%	29.85%	16.04%
SMOAP-16	2892	62	OM	OM	2038	62	OM	OM	29.53%	OM	OM
SMOAP-17	3756	80	OM	OM	2552	80	OM	OM	32.06%	OM	OM
SMOAP-18	4572	96	OM	OM	3133	96	OM	OM	31.47%	OM	OM
SMOAP-19	5060	105	OM	OM	3519	105	OM	OM	30.45%	OM	OM
SMOAP-20	5185	110	OM	OM	3561	110	OM	OM	31.32%	OM	OM

Another issue that should be considered in the preprocessing stage is the date consistency. Recall that two orders o_i^P and o_j^S can be matched if and only if $G_j^S \subseteq G_i^P$ and $T_i^P \cap T_j^S \neq \phi$. Each matching can be regarded as an edge connecting two nodes (demand orders, supply orders) in a weighted bipartite graph. Therefore, the potential matchings that violate the date consistency are deleted from the graph and no longer checked during the local search process. After deleting inconsistency edges, the nodes whose degree becomes zero are further removed. Note that suppliers are allowed to submit orders with their own available date. This rule makes it possible for unselected purchasers to adjust their orders in the second round.

The reduction does take effect, which can be seen in Table 2. We report the reduction on suppliers, purchasers, variables and constraints. The number of variables is mostly reduced by 30% to 40%, which has shown a relatively consistent trend with suppliers. As for constraints, they have a reduction rate of 15% to 25%, around 12% lower than variables. However, the number of purchasers remains unchanged for the reason that all the demand orders have a series of corresponding union set in the benchmarks.

Divide-and-conquer is an efficient method to deal with large-scale optimization problems. As the data volume of this trading platform may be large, we adopt this idea to decompose original problems into several subproblems that are easier to solve. In SMOAP, $|P|$ components are naturally divided in accordance with the number of purchasers. For each subproblem $Group_p$, the algorithm tries to obtain one optimal solution with related supply orders. Eventually, these solutions are integrated into one allocation plan as the final result. It should be noted that some supply orders can be divided into different groups for the reason that purchasers may have common interest in the same commodity set. However, such kind of orders can be allocated only once.

As shown in Algorithm 1, the *Preprocessing* procedure is efficient with a time complexity of $O(|P| * |S|)$ and executed only once. The output of this process is delivered as the input of tabu search.

Algorithm 1 Preprocessing

$Group = \{Group_1, ..., Group_{|P|}\} = \phi$;
for all $p \in P$ **do**
  for all $s \in S$ **do**
    if $G_s^S \subseteq G_p^P$ **then**
      if $T_p^P \cap T_s^S \neq \phi$ **then**
        $Group_p = Group_p \cup s$;
      else
        delete $< p, s >$;
      end if
    end if
  end for
  if $degree(p) = 0$ or $\bigcup_{i \in Group_p} G_i^S \neq G_p^P$ **then**
    delete p and $Group_p$;
  end if
end for
return $Group$;

2.3. Neighborhood structures

When we talk about local search methods, the most important component is the neighborhood structures, which must be carefully designed. Given a current solution X, the neighborhoods of X are defined as $N_i(X)$, where different values of i represent different types of neighborhood solutions. At each iteration, the best solution among $N_i(X)$ is selected to be the new solution in the next round, ensuring the searching direction towards the space of interest. The scope of each neighborhood structure determines the convergence, while the diversity of these neighbors influences the effectiveness positively and helps prevent the local optimal. Therefore, we propose two neighborhood structures $N_1(X), N_2(X)$ to be the candidate pools. As the decomposition method is used through the whole process, $N_1(X), N_2(X)$ are adopted within subproblems. Specific formulations and explanations are stated as follows.

For a solution $X = \{x_1, ..., x_i, ..., x_j, ..., x_n\}$, if $x_i = 1$ and $x_j = 1$, then $remove(x_i, x_j) = \{x_1, ..., \overline{x_i}, ..., \overline{x_j}, ..., x_n\}$, where $\overline{x_i}$ is the opposite variable of x_i. Note that this two-step remove can also represent the condition of an one-step remove via $remove(x_i, \phi)$. After this operation, X becomes infeasible, allowing other combination of orders to be explored. For an infeasible solution $X = \{x_1, ..., x_i, ..., x_n\}$, $add(x_i)$=$\{x_1, ..., \overline{x_i}, ..., x_n\}$, if $x_i = 0$. In this step, one variable with the value of 1 in $\overline{X} = \{\overline{x_1}, ..., \overline{x_i}, ..., \overline{x_n}\}$ is selected and added into the solution. Based on this, $repair(X)$=$\{\bigcup_{i=1}^{mv} add(x_i)\}$, where mv is the minimum number of moves to make X feasible. As the tabu list exists, $repair(X)$ tries to add long-term forgotten orders into the solution first, which contributes to the diversity of the local search. Therefore, two kinds of neighborhoods are defined as $N_1(X) = \{repair(remove(x_i, \phi))\}$ and $N_2(X) = \{repair(remove(x_i, x_j))\}$.

In SMOAP, the neighborhood $Neighbor(X)$ is defined by two operators: *remove*(x_i, x_j) and *repair*(X). The local search will move along superior neighborhoods for each step to get optimal. The effects of these two neighbors are verified in the experiment part.

2.4. Tabu Search

Tabu search (TS) is a metaheuristic method that systematically impose and release constraints to guide the local search procedure across the local optimal (Glover & Marti, 1997). The two key elements of TS are tabu list and tabu tenure. If a potential solution has been previously visited within a certain period, it is marked as "tabu" and added to the tabu list so that the algorithm does not consider that possibility repeatedly. When its tabu tenure is over, it returns to the available state and can be selected again. There are two ways when designing the tabu list structure, that is, solution-based tabu lists and operation-based tabu lists. We employ the operation-based tabu list due to its efficiency and easy-implementation. Different from solution-based tabu lists which will check the solution state during each local search, the operation-based lists only taboo certain operations (e.g., *remove* in our LS) performed on one order recently. TS is considered to be an efficient local search for combinational optimization problems, as it can accept worse moves if all other improving moves are forbidden in the tabu list.

The tabu search used in this paper is stated in Algorithm 2. Two neighborhoods, $N_1(X) = \{repair(remove(x_i, \phi))\}$ and $N_2(X) = \{repair(remove(x_i, x_j))\}$, are explored to select the most profitable solution (line 1). Every time we remove or add orders, the tabu list should firstly be checked to make sure that the move is allowed (line 2). If a better solution X' is generated from $N(X)$, update the current solution X and continuously search from this new solution (line 3-5). After the move, these orders are pushed into the tabu list to avoid cyclic search (line 7). Furthermore, an update of the tabu tenure is followed (line 8).

Algorithm 2 Tabu search procedure

Input: Current solution X
Output: Best solution X found
1: $X' = N(X)$ with the greatest *score*;
2: **if** X' is not tabu **then**
3: **if** $benefit(X) > benefit(X')$ **then**
4: $X = X'$;
5: **end if**
6: **end if**
7: Update $Tabulist$;
8: Update $Tabutenure$;
9: return X;

2.5. Some discussions about local optimal

For local optimal challenges, two perturbation levels are adopted in our algorithm, from tiny to violent adjustment.

(1) When deciding the next step towards the best solution, heuristics may work well, yet potentially cause the premature convergence. In this case, a random move is employed with a low probability of $1 - rnd$, and the best (greedy) move with probability rnd during local search process. This is called a tiny perturbation level because it only changes one-step move of the algorithm.
(2) A restart strategy is applied to escape from local optimum and to restart the

search in other areas of the search place. If the best solution has not been continuously improved for rs iterations in one subproblem, the local search will reinitialize from a randomly generated solution. The scope of restart is within this subproblem. There is no need to restart all the subproblems, especially the ones without sufficient searching. This is called a violent perturbation level because it changes the current solution of one subproblem.

References

Glover, F., & Marti, R. (1997). Tabu search. *General Information*, *106*(2), 221-225.