Mathematical Biosciences and Engineering

http://www.aimspress.com/journal/mbe

MBE, 19(3): 3202–3222.
DOI: 10.3934/mbe.2022148
Received: 31 August 2021
Revised: 27 December 2021
Accepted: 13 January 2022
Published: 20 January 2022

Research article

A hybrid ant colony algorithm for the winner determination problem

Jun Wu[1], Mingjie Fan[2], Yang Liu[1], Yupeng Zhou[1,*], Nan Yang[3] and Minghao Yin[1,4,*]

[1] Information Science and Technology, Northeast Normal University, Changchun, China
[2] School of Science, Beijing University of Posts and Telecommunications, Bejing, China
[3] CHEARI Certification & Testing Co., Ltd., Beijing, China
[4] Key Laboratory of Applied Statistics of MOE, Northeast Normal University, Changchun, China

* **Correspondence:** Email: ymh@nenu.edu.cn, zhouyp605@nenu.edu.cn.

Abstract: Combinatorial auction is an important type of market mechanism, which can help bidders to bid on the combination of items more efficiently. The winner determination problem (WDP) is one of the most challenging research topics on the combinatorial auction, which has been proven to be NP-hard. It has more attention from researchers in recent years and has a wide range of real-world applications. To solve the winner determination problem effectively, this paper proposes a hybrid ant colony algorithm called DHS-ACO, which combines an effective local search for exploitation and an ant colony algorithm for exploration, with two effective strategies. One is a hash tabu search strategy adopted to reduce the cycling problem in the local search procedure. Another is a deep scoring strategy which is introduced to consider the profound effects of the local operators. The experimental results on a broad range of benchmarks show that DHS-ACO outperforms the existing algorithms.

Keywords: ant colony algorithm; local search; combinatorial auction; winner determination problem; tabu search

1. Introduction

The winner determination problem (WDP) in combinatorial auction aims to determine a bid-winning scheme according to the combination of items submitted by various bidders as well as their bids. The object of WDP is to maximize the revenue of auctioneers, where the revenue is measured by the total price of the selected bids. Compared with traditional auction mechanisms, WDP can improve the efficiency of the auction process and reduce the number of failure bids [1]. WDP is widely used in cloud computing [2], online reverse auctions [3,4], spectrum license sales by America's Federal Communications Commission (FCC)*, e-commerce [5], wireless network [6], production management [7],

*`http://wireless.fcc.gov/auctions`

game theory [8], and multi-agent system resource allocation [9, 10], etc. Solving the winner determination problem effectively can improve the utilization of items, which is conducive to sustainable development. However, the process of determining the winning criterion is NP-hard problem [11]. How to solve WDP effectively has become a hot topic in the field of combinatorial optimization.

Nowadays, the memetic algorithm which integrates the population-based method and the heuristic or meta-heuristic method performs well on many combinatorial optimization problems [12–15]. The population-based methods, which can expand the search space of the problem, include genetic algorithm [16], ant colony algorithm [17], bee colony algorithm colony [18], and so on.

Among the population-based methods, the ant colony algorithm (ACO), which uses the pheromone model and heuristic information of the problem to construct solutions in a probabilistic way [19]. Pheromone trails are the result of a learning mechanism that tries to identify the solution components that, when appropriately combined, lead to high-quality solutions. However, as well known, ACO and other population-based methods have strong robustness, but its convergence speed is slow and is easy to fall into the local optima.

Therefore, in many combinatorial problems, the heuristic or meta-heuristic methods are essential for obtaining competitive solutions, which can be used to prevent the search from premature convergence, including local search [20], tabu search [21], and simulated annealing [22], etc. Tabu search (TS) is a widely studied heuristic method for its distinguishing features of adaptive memory and responsive exploration [23]. This method maintains a short-term memory of the specific changes in some recent moves within the search space, and prevent future moves from undoing those changes. Due to its capacity of searching effectively, TS has been applied to solve various combinatorial problems, such as minimum weight vertex independent dominating set problem [24], the multi-compartment vehicle routing problem [25], and multi-AGV routing problem [26].

Consequently, in this paper, we focus on improving the basic ACO by blending an effective local search procedure for solving WDP. To be specific, a hash tabu method is proposed to prevent the revisiting of those crucial solutions marked in the search history. As the searching direction is guided by the scoring function of the local search, we further propose a deep scoring strategy, which not only considers the environment of incumbent solutions, but also takes subsequent operations into account. We present the computational results of the proposed algorithm on 44 basics and 13 extended benchmark instances commonly used in the literature, and compare our results with those of the state-of-the-art algorithms for WDP. The results indicate that DHS-ACO is quite efficient on basic benchmark instances and outperforms other competitors on extended test suits. Furthermore, the verification experiments on different components of DHS-ACO are conducted to verify the effectiveness of the hash tabu strategy, the deep scoring strategy, the local search, and the ant colony algorithm.

The remainder of this paper is organized as follows. In the next section, we introduce the related work of practical algorithms for solving the WDP. In Section 3, we introduce some preliminary knowledge about both DHS-ACO and WDP. Section 4 describes the details of the components in the local search. Based on the above, DHS-ACO is presented. Then, we carry out comparison and verification experiments to evaluate DHS-ACO in Section 5. In the last section, we conclude the paper and introduce some future work.

2. Related work

In recent years, WDP and its variants have been well studied due to their extensive applications. Vangerven et al. [27] adapted the winner determination problem for geometrical combinatorial auctions. Then, a new subclass of WDP, i.e., the network winner determination problem (NWDP), was proposed in [28], which characterized different problems in NWDP class and analyzed their computational complexity. Afterwards, Remli et al. [29] addressed the winner determination problem for TL transportation procurement auctions under uncertain shipment volumes and uncertain carriers' capacity. It extended an existing two-stage robust formulation proposed for WDP with uncertain shipment volumes. In the same year, Qian et al. [30] conducted a research on winner determination of loss-averse buyers with incomplete information in multi-attribute reverse auctions for clean energy device procurement. It was the same team that further studied a revised winner determination problem with disruption risk of bidders for a fourth party logistics (4PL) provider to purchase transportation services via combinatorial reverse auction [31]. In 2020, Lee et al. [32] resolved the integration difficulty between scheduling and routing aspects of the multiple automated guided vehicle (AGV) problem that were modelled by the winner determination problem.

For the WDP studied in this work, a number of exact algorithms have been proposed. Fujishima et al. [33] proposed a method, which is guaranteed to be optimal, to reduce the running time by structuring the search space. Nisan [34] suggested an approach based on linear programming (LP) and proved that the LP approach finds an optimal allocation if and only if prices can be attached to single items in the auction. Leyton-Brown et al. [35] proved the correctness of a branch-and-bound algorithm, which incorporates a specialized dynamic programming procedure. Sandholm and Suri [36] presented a more sophisticated search algorithm, including several technologies, for determining winners in many generalizations. Günlük and Ladányi [37] presented a column generation-based algorithm to solve the WDP given in the XOR-of-OR language and a methodology to generate realistic test problems. Escudero et al. [38] proposed a new and tighter formulation of WDP and new valid inequalities; then they presented a branch-and-cut algorithm which shows its efficiency in a big number of instances.

However, the exact methods can obtain the optimal solution of WDP, while they may fail to return a high-quality solution within a reasonable time or for large-scale of instances. Therefore, Hoos and Boutilier [39] proposed a stochastic local search algorithm (named Casanova), which added a bid to the current solution by considering bids' scores and history information. Based on the basic framework of the simulated annealing algorithm, Guo et al. [40] introduced the SAGII algorithm with three hybrid moving operators, i.e., the branch-and-bound move, the greedy local search move and the exchange move. Experimental results showed that SAGII performed better than Casanova algorithm. A new strategy was proposed in [41], in which a bid was added randomly with probability p at each local iteration, and the bid with maximum profits was added into the solution with probability $1 - p$. Experimental results showed that this strategy had a remarkable effect on the improvement of the algorithm. Wang [42] proposed a new super-heuristic algorithm (named SHH) by combining the selection function and random strategy. Based on the framework of the ant colony algorithm, Dowlatshahi et al. [43] performed a multi-neighborhood local search algorithm. By combining a stochastic local search component with a specific crossover operator, Boughaci et al. [44] presented a memetic algorithm for the winner determination problem. An efficient local search algorithm (named abcWDP) was proposed in [45], which used a pattern monitoring strategy to guide the search. Experimental results illustrated

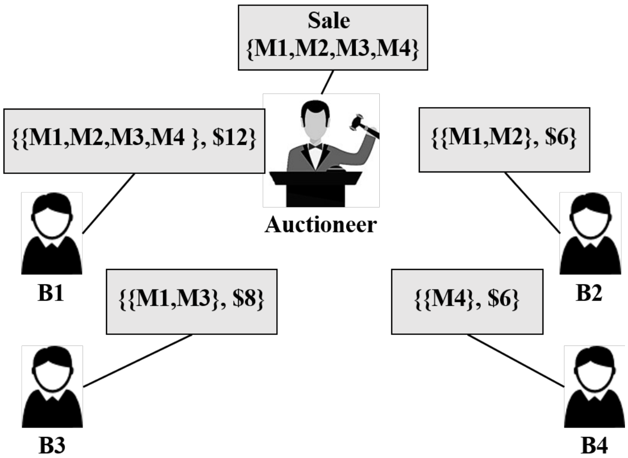

Figure 1. A scene of WDP.

that this algorithm was the state-of-the-art heuristic algorithm for WDP. In addition, Lin et al. [46] raised to transform the combinatorial auction optimization problem into an unconstrained integer programming problem, and proposed a DCM algorithm to solve it.

3. Preliminaries

3.1. Definition and notations

Before we give the formulation of WDP, some notations are firstly introduced. Suppose that the collection of items auctioned is $MS = \{M_1, M_2, ..., M_t\}$, where t is the quantity of items. The set of bids provided by all bidders is $BS = \{B_1, B_2, ..., B_n\}$, where n is the number of bids. Each bid $B_j = \{S_j, P_j\}$ has two properties: S_j is a non-empty subset of MS that represents the combination of items to bid B_j, while P_j is the bid price of B_j and $P_j \geq 0$. Assume that a Boolean array $X = \{x_1, x_2, ..., x_n\}$ is a feasible solution of WDP, where $x_j \in \{0, 1\}$. $x_j = 1$ means that the bid B_j is selected by the auctioneer, i.e., B_j is the winning bid, i.e., B_j is selected in the solution; otherwise, $x_j = 0$ indicates that B_j is not selected by the auctioneer, i.e., it is a losing bid. Given above, the constraint of WDP is described as follows:

$$\max f(x) = \sum_{j=1}^{n} P_j x_j \tag{3.1}$$

$$\text{subject to: } \sum_{j=1}^{n} a_{ij} x_j \geq 1 \quad i \in \{1, 2, 3, ..., t\}, a_{ij} \in \{0, 1\} \tag{3.2}$$

where $a_{ij} = 1$ represents $M_i \in S_j$, otherwise, $M_i \notin S_j$, $i \in [1, t]$. Constraint 3.2 guarantees that each item can be allocated at most once. Therefore, Eq (3.1) aims to maximize the auctioneer's revenue by selecting a set of winning bids.

In order to make the auction process clearly, Figure 1 shows the composite auction scene. Note that two bids containing at least a same item are considered as conflicting bids, which cannot exist simultaneously in the solution. For example, in this scene, B_2 and B_3 conflict with each other as they both contain M_1. All feasible solutions of WDP are composed of different non-conflicting bids. Therefore, all feasible solutions in this example are : $C_1 = \{\{B_1\}, 12\}$, $C_2 = \{\{B_2\}, 6\}$, $C_3 = \{\{B_3\}, 8\}$, $C_4 = \{\{B_4\}, 6\}$, $C_5 = \{\{B_2, B_4\}, 12\}$, $C_6 = \{\{B_3, B_4\}, 14\}$. Among all feasible solutions, C_6 is the optimal solution, as it brings maximum benefit to the auctioneer. That is, when the auctioneer selects B_3 and B_4, the auctioneer will get the highest income \$14.

3.2. Ant colony algorithm

Ant colony algorithm (ACO) constructs the solution step by step in the way of selecting probability. For WDP, the specific calculation method of selection probability can be calculated as follows. An initial solution C is given to store all winning bids. Let $FB = \{B|CB(B) \cap C = \emptyset\} \cap \{BS \backslash C\}$ collect all bids without any conflict bids in C, where $CB(B)$ represents all bids that conflict with B in BS. Given the candidate bid set FB, the chosen probability of bid $B_j \in FB$ is:

$$p_j = \frac{(\tau_j)^\alpha \times (\eta_j)^\beta}{\sum\limits_{B_k \in FB} (\tau_k)^\alpha \times (\eta_k)^\beta} \tag{3.3}$$

where α and β represent the influence factors of pheromone and heuristic information of the ant colony algorithm respectively, and η_j is the heuristic factor for solving WDP. Considering the goal of WDP, which is to maximize the auctioneer's revenue, we use the bid price P_j to replace the heuristic factor η_j.

The basic process of ACO for solving WDP is as follows: firstly, initialize the pheromone vector of each element $\tau_j = pvi$, where pvi is the initial pheromone concentration. Then, the algorithm enters the search phase. At each iteration, every ant chooses bids from FB according to Eq (3.3). Then, the colony pheromones are updated as Eq (3.4) shows.

$$\tau_j = (1 - \rho) \times \tau_j + \Delta \times x_j, \forall j \in \{1, 2, ..., n\} \tag{3.4}$$

where $\rho \in [0, 1]$ represents the volatility of the pheromone. Δ is defined as:

$$\Delta = \frac{Fitness(C)}{\sum_{k=1}^{n} P_k} \tag{3.5}$$

where $Fitness(C)$ represents the fitness of solution C, calculated as:

$$Fitness(C) = \sum_{B_j \in C} P_j \tag{3.6}$$

After the pheromone updating process is completed, ACO judges whether the stop condition is met. If not, the algorithm enters the next iteration. Otherwise, ACO ends in outputting the best solution ever found.

4. Hybrid ant colony algorithm for WDP

In this section, the proposed algorithm named DHS-ACO is introduced. Firstly, the main strategies of the local search are described with detailed explanations. Then, the whole profile of the local search is introduced. Finally, the framework of DHS-ACO is given.

4.1. Local search procedure

Local search is an effective method to enhance the quality of the newly generated solutions by exploiting their surroundings. In this section, two key methods for the local search are introduced, including the hash tabu method and the deep scoring strategy. Then, the description of the move operator used in the local search is given. Finally, the local search framework is summarized.

4.1.1. Hash tabu method

To avoid the repeated visiting of the same solution in the local search procedure, the hash tabu strategy is designed to efficiently determine whether the current solution has been visited. Given a solution C and a prime number pr, the hash value corresponding to solution C is defined as:

$$hash(C) = (\sum\nolimits_{B_i \in C} 2^{i-1}) \mod pr, i \in \{1, 2, ..., n\} \tag{4.1}$$

The hash tabu strategy employs a Boolean hash table HT, whose length is pr. The larger the pr value is, the longer the hash table length is, and the less the possibility of hash clash is. Specifically, the method uses a hash table to judge whether the algorithm returns to the visited solution again. In other words, $HT_h = 1$ indicates that the solution with a hash value of h has been marked in the search history, while $HT_h = 0$ indicates that the solution has not been marked. At the beginning of the algorithm, each element in HT is set to 0, and a hash secondary array H with length n (n is the number of bids) is created, in which each element $H_i = 2^{i-1} \mod pr$. In the initialization, we first set $H_1 = 2^0$ (mod pr) = 1, and the subsequent elements are calculated according to the following equation:

$$H_i = 2H_{i-1} \mod pr, i \in \{2, 3, ..., n\} \tag{4.2}$$

When adding or removing the bid B_i to the solution C, the corresponding hash value of C is updated according to the following equations:

$$hash(C \cup \{B_i\}) = [hash(C) + H_i] \pmod{pr} \tag{4.3}$$

$$hash(C \setminus B_i) = [hash(C) + pr - H_i] \pmod{pr} \tag{4.4}$$

Then, the specific usage of the hash tabu strategy is introduced. In the local search procedure, if the quality of the local search solution decreases in the previous step, while improved in the current step, one of the following situations which will be executed determined by the value of HT_h, and $h = hash(C)$.

1) If $HT_h = 1$, the last move will be cancelled with probability ph. In this case, the selected bid B_{win} will be removed and prohibited from being added to the current solution until any of its conflicted bids are removed from C. With probability $1 - ph$, the algorithm will rebuild C according to Eq (3) and restart the local search to avoid repeated visiting.
2) Otherwise, the algorithm will set $HT_h = 1$ and continue to search.

4.1.2. Deep scoring strategy

The deep scoring strategy is applied in the bid selection of the local search. The main idea of this mechanism is that if $FB \neq \emptyset$, then a bid with the largest $SScore$ value is selected from FB to be added to the current solution. FB stores those bids that are not conflicted with the current solution and $SScore$ is calculated as follows:

$$SScore(B_i) = Score(B_i) - Subscore(B_i) \tag{4.5}$$

where $Score()$ reflects the fitness value change of C after B_i is added:

$$Score(B_i) = P_i - \sum\nolimits_{B_j \in C \cap B_j \in CB(B_i)} P_j \tag{4.6}$$

Clearly, if $B_i \in FB$ and $CB(B_i) = \emptyset$, the time complexity of the Eq (4.6) will reduce to $O(1)$. The $Score()$ reflects the greedy degree of the local search, because it only considers the impact of adding one bid to the current solution and does not consider the impact on subsequent local search operators. Regarding this, we introduce the $Subscore()$ evaluation function to estimate the loss of FB set after adding one bid to C, which is calculated as follows:

$$Subscore(B_i) = \sum\nolimits_{B_j \in FB \cap CB(B_i)} \frac{P_j}{|S_j|} \tag{4.7}$$

On the one hand, the deep scoring strategy ensures that the local search can choose those bids in FB first. Since there are no conflict bids between FB and C, it is unnecessary to conduct conflict detection process and remove the conflicting bid when a bid in FB is added. Therefore, giving priority to select bids in FB will significantly reduce the selecting time. On the other hand, when selecting bids in FB, the deep scoring strategy uses $SScore()$ to comprehensively consider the impact on the solution and the impact on FB loss, which makes the bid selection more reasonable.

4.1.3. Move operator

Algorithm 1: Move

Input: the current solution C,configuration checking table CC
Output: A feasible solution of given auction

if $FB \neq \emptyset$ **then**
    select $B_{pick} \in FB$ with the greatest SScore ;
else
    $rn \leftarrow$ find a number form [0,1] randomly;
    if $rn < p$ **then**
        $L = \{B | B \in BS \backslash C \cap CC(B) = 1\}$;
        $B_{pick} = OldestGoodBid(L)$;
    else
        $B_{pick} = RandomBid(BS \backslash C)$;
add B_{pick} into C;
remove all conflicting bids from C except B_{pick};
return C;

The local search procedure exploits the neighborhoods of the current solution by performing the move operator, whose pseudo-code is shown in Algorithm 1.

In algorithm 1, the input data includes the current solution C and the configuration checking vector CC. CC was proposed by [45], whose updating method is as follows: at the beginning of the local search, all elements of CC are initialized to 1. When a bid is removed from the current solution C, the bid CC value is set to 0, and all bids that conflict with the bid CC value are set to 1.

The algorithm first detects whether FB is empty. If $FB \neq \emptyset$, the bid with the largest $SScore$ will be selected into C. Otherwise, the following steps are executed:

1) In the case of probability p, the algorithm puts the bid with $CC = 1$ into a temporary set L and calls $OldestGoodBid()$. The $OldestGoodBid()$ function performs the following operations: if $L = \emptyset$, then the bid to be added into C will be randomly selected from $BS \backslash C$. Otherwise, c $T = \{B | B \in L \cap Score^* - Score(B) \leq \delta \cdot std_B\}$, where $Score^*$ is the maximum score of all bids in L, δ is the parameter that controls the scale of T, and std_B is the standard deviation of all bids' prices. Finally, the bid with the longest iterations outside C will be selected into C.
2) Randomly select a bid from $BS \backslash C$ with probability $1 - p$ and add it into C to enhance the diversity of the algorithm.

After completing any of the above cases, the algorithm removes all bids that conflict with B_{pick} in C and returns C.

Algorithm 2: Local_search

```
Input: current solution C
Output: the best solution C* found
C* = C, N = 0, lastStepImproved = 1;
init(CC);
while N = max_no_improved do
    oldC = C, C = Move(C);
    if Fitness(C) ≤ Fitness(oldC) then
        lastStepImproved = 0, N = N + 1;
    else
        N = 0, C* = best(C, C*);
        if lastStepImproved = 0 then
            h = hash(C);
            if HT(h) = 1 then
                if rand(0, 1) < ph then
                    C = oldC, CC(B_pick) = 0;
                else
                    C = newAnt(), init(CC), N = 0;
            else
                HT(h) = 1;
        lastStepImproved = 1;
return C*;
```

Based on the ideas proposed above, the pseudo-code of the local search is given in Algorithm 2. At the beginning, the local search procedure initializes the best solution C^* as the current solution C, the non-improved number of the fitness $N = 0$, and the boolean variable $lastStepImproved = 1$, presenting the condition that the fitness of C has been improved after the last move (line 1). Moreover, all elements in CC are initialized to 1 to allow the case that all bids are enabled to be added (line 2). Then, the algorithm iterates until N reaches the limit $max\_no\_improved$ (lines 3–25). At each iteration of the loop, the $Move$ operator is applied to exploit the neighborhood of C (line 5). If the $Move$ operator failed to find a better solution, the algorithm will set $N = N + 1$ and mark $lastStepImproved$ as 0 (lines 6–8). Otherwise, the algorithm will refresh $N = 0$ (line 10), and attempt to update C^* (line 11). Afterwards, the hash tabu strategy described in Section 4.1.1 is executed (lines 12–23). Finally, the best solution C^* found during the search will be returned (line 25).

4.2. The DHS-ACO algorithm

In this section, we design a hybrid ant colony algorithm called DHS-ACO to solve WDP. The pseudo-code of the DHS-ACO algorithm is given in Algorithm 3.

At the start of the algorithm, DHS-ACO initializes the pheromone vector τ, the hash table HT, the hash auxiliary array H and the best solution $bestC$ (lines 1–2). In each subsequent iteration, the algorithm firstly builds each ant by using the pheromone (line 5). Then, the local search procedure is adopted to exploit the neighborhood of the new ant (line 6). After all ants have finished the local search, DHS-ACO updates the pheromone vector τ (line 8). The loop will end when the running time exceeds the time limit (line 3). Finally, DHS-ACO outputs the best solution $bestC$ (line 9).

Algorithm 3: DHS-ACO

Input: bids set BS, items set MS

Output: the best solution found $bestC$

```
1 init(τ, H, HT);
2 bestC = ∅;
3 while stopping criteria is not satisfied do
4     for each ant do
5         C = newAnt(τ);
6         C = Local_search(C);
7         bestC = best(C, bestC);
8     τ = update_pheromone();
9 return bestC;
```

4.3. Time complexity of DHS-ACO

In this section, the complexity of the key components of DHS-ACO are calculated, including the ant construction, the pheromone updating, and the local search procedure. For a brief instruction, some notations are recalled: $AntNum = p_a$, $|FB| = s_f$.

The ant construction process builds an ant by selecting bids from FB one by one until $FB = \emptyset$. The selected bids are determined within $O(s_f)$ time. Since $|C| \leq n$ and $s_f \leq n$, the time complexity of constructing one ant is no more than $O(n^2)$. As for the pheromone updating, all n elements of τ

are updated according to the fitness of all p_a ants. Therefore, it takes $O(np_a)$ time. In the local search procedure, it takes $O(n)$ for the Move operator in each iteration. Besides, at most $O(n^2)$ time would be taken if the hash tabu strategy decides to rebuild the current solution. Thus, the worst case time complexity of one iteration of the local search is $O(n + n^2)$, i.e., $O(n^2)$.

5. Experimental evaluation

To show the efficiency of the proposed algorithm, we carry out extensive experiments in this section. Firstly, the benchmarks used in the experiments are introduced. Then, we compare DHS-ACO against a number of state-of-the-art competitors, including abcWDP [45], DCM [46], MA [44] and HBHSA [47], to show the algorithmic efficiency. Furthermore, the strategy verification experiments are performed to verify the effectiveness of the hash tabu method, the deep scoring strategy, the ant colony framework and the local search procedure proposed in this paper.

5.1. Benchmarks

The benchmarks used in this work are selected from a set of LG benchmarks proposed in [48]. As shown in Table 1, it contains 500 instances and is divided into five classes according to the number of bids and items. All classes are named as REL_X_Y, where X represents the number of bids and Y is the number of items. Each class contains 100 examples. We refer to [44–47] and only select 44 samples from them as the basic benchmarks.

In addition, due to the small performance gap between DHS-ACO and other competitors on the basic benchmarks, we use the CATS platform [49] to generate a larger set of 13 extended instances, to further compare the performance of DHS-ACO and the state-of-the-art solvers on large-scale instances. Compared with the basic benchmarks, the number of bids and items increases significantly, which will become the obstruction for algorithms. Note that abcWDP performs best among the competitors and thus is chosen as the comparison on extended benchmarks. The extended benchmarks are listed in Table 2. All instances used in this work can be found on the website †.

Table 1. The message of the basic benchmarks.

Class	Instances	Selected instances
REL_1000_500	in101 - in200	in101 – in110
REL_1000_1000	in201 - in300	in201 – in210
REL_500_1000	in401 - in500	in401 – in410
REL_1500_1000	in501 - in600	in501 – in504
REL_1500_1500	in601 - in700	in601 – in610

† https://github.com/wujunzero/WDP

Table 2. The message of extended benchmarks.

Name	Number of bids	Number of items
CATS_2000	2000	2005
CATS_2500	2500	2504
CATS_3000	3000	3004
CATS_3500	3500	3504
CATS_4000	4000	4001
CATS_4500	4500	4504
CATS_5000	5000	5002
CATS_5500	5500	5501
CATS_6000	6000	6000
CATS_6500	6500	6505
CATS_7000	7000	7005
CATS_7500	7500	7505
CATS_8000	8000	8005

5.2. *Experiments and analysis*

The DHS-ACO is implemented in C++. For DHS-ACO and abcWDP, all experiments are conducted 10 times on a computer with Intel(R) Xeon(R) CPU E7-4820 v4 @ 2.00 GHz, 64 GB. The experimental results of DCM and MA are taken from [46], while the results of HBHSA are taken from [47]. The operating environment is Intel i3-2330@2.2 GHz, Intel Pentium IV@2.8 GHz and 3.4 GHz AMD processors, respectively. Obviously, their CPU base frequency is higher than that of the experimental environment in this paper (2.00 GHz).

5.3. *Parameter settings*

We use the automatic tuning tool irace [50] to tune the parameters. We randomly select 50 examples from all the examples as the training set. The parameter tuning process is budgeted to run 2,000 times, each with a budgeted run time of 1000 seconds. The final values obtained by irace are shown in Table 3. The results of each experiment will be introduced and analyzed in the following part.

Table 3. Parameter setting of DHS-ACO.

Parameters	Description	Ranges	Final values
$AntNum$	number of ants	{6, 8, 10, 12}	10
pvi	initial pheromone concentration	{5, 10, 15, 20}	10
ρ	volatility of pheromone	$\{0, 0.1, \ldots, 0.9\}$	0.1
α	influence factors of pheromone	$\{0, 1, \ldots, 5\}$	1
β	influence factors of heuristic information	$\{0, 1, \ldots, 5\}$	1
δ	parameter that controls the scale of establish set	$\{0.1, 0.2, \ldots, 0.9\}$	0.5
p	allow probability of adding a bid into current solution	$\{0.91, 0.92, \ldots, 0.99\}$	0.95
$max\_no\_improved$	maximum non-improved steps	{80, 90, 100, 110}	100
pr	length of the Boolean hash table	$\{10^9, 10^9 + 1, \ldots, 10^9 + 10\}$	$10^9 + 7$
ph	cancel probability of a move operator	$\{0.90, 0.91, \ldots, 0.99\}$	0.95

5.4. Comparison with state-of-the-art competitors

The experimental results of DHS-ACO and its competitors on basic benchmarks are presented in Tables 4–8. The columns named f and t stand for the average fitness and the average convergence time, respectively. Table 9 summarizes the average of the fitness ("*avg_f*") and the total time ("*total_t*", in seconds) of all algorithms in each class of basic benchmarks shown in Tables 4–8. Figure 2 shows the average convergence time ("*avg_t*") and the number of reaching the optimal fitness among all reference algorithms ("*best_num*") of the best solution obtained by each compared algorithm on all 44 basic instances.

Since the performance gap between DHS-ACO and abcWDP is small on basic benchmarks, we use the extended benchmarks to compare the performance of DHS-ACO and abcWDP. As shown in Table 10, compared with abcWDP algorithm, DHS-ACO algorithm is superior to abcWDP on the average objective function values obtained, which demonstrates that DHS-ACO performs better than abcWDP in solving large-scale instances. "Total" in the last row means the execution time of the tested instances in seconds.

Therefore, according to the results in Tables 4–9 and Figure 2, DHS-ACO algorithm is obviously superior to HBHSA, DCM and MA regarding the performance on each group of the basic benchmarks. Compared with the reference algorithms, DHS-ACO can obtain the same fitness values on all basic benchmarks, owing to the exploration of the heuristic method. And the average convergence time of DHS-ACO is faster and performs 1–2 orders of magnitude faster than the competitors, which reflects the effectiveness and efficiency of DHS-ACO on basic benchmarks. Moreover, for the extended benchmarks (the results of the comparison are shown in Table 10), due to the large-scale of these benchmarks, it is difficult for DCM, MA and HBHSA to solve them, while DHS-ACO obtains the better solutions by reducing the convergence time in each search iteration. Note that DHS-ACO can also find the better solutions on all extended benchmarks than abcWDP, although it requires more execution time than abcWDP in total. It is because of DHS-ACO explored enlarged feasible search space, benefited of the population-based framework.

Table 4. Results on subset of REL_1000_500.

Instances	DHS-ACO		abcWDP		DCM		MA		HBHSA	
	f	t	f	t	f	t	f	t	f	t
in101	**72724.62**	**2.41**	**72724.62**	5.55	67330.25	40.46	67101.93	129.62	67973.71	59.51
in102	**72518.22**	**4.19**	**72518.22**	10.31	70186.90	43.67	67797.61	132.18	70706.86	58.16
in103	**72129.50**	**2.35**	**72129.50**	12.51	67496.73	45.38	66350.99	133.34	69151.79	58.28
in104	**72709.65**	**3.77**	**72709.65**	62.78	69791.24	44.60	64618.41	135.14	71184.80	59.24
in105	**75646.13**	1.78	**75646.13**	1.35	69274.29	47.05	66376.83	153.96	72725.20	62.98
in106	**71258.61**	**1.40**	**71258.61**	1.49	65110.89	42.82	65481.64	140.96	66461.43	57.67
in107	**69713.40**	**1.32**	**69713.40**	1.93	67026.55	40.11	66245.70	146.40	68476.54	56.38
in108	**75813.21**	3.32	**75813.21**	1.54	73357.63	46.34	74588.51	161.03	72729.34	58.18
in109	**69475.90**	**1.16**	**69475.90**	1.36	64548.53	41.34	62492.66	144.71	66023.87	59.74
in110	**68295.29**	**3.72**	**68295.29**	3.73	65547.86	45.32	65171.19	149.01	66405.36	59.11

Table 5. Results on subset of REL_1000_1000.

Instances	DHS-ACO		abcWDP		DCM		MA		HBHSA	
	f	*t*	*f*	*t*	*f*	*t*	*f*	*t*	*f*	*t*
in201	**81557.74**	0.22	**81557.74**	0.17	78856.30	70.34	77499.82	98.26	80079.58	80.92
in202	**90708.13**	**1.37**	**90708.13**	3.93	88850.75	79.42	90464.19	106.68	90490.79	85.73
in203	**86239.21**	0.99	**86239.21**	0.89	82551.15	75.48	86239.21	102.28	84491.86	82.22
in204	**87075.43**	0.78	**87075.43**	0.55	83666.49	72.00	81969.05	97.40	85057.33	84.55
in205	**86515.95**	**0.92**	**86515.95**	1.87	84130.23	71.92	82469.19	91.26	85422.67	116.05
in206	**91518.96**	**0.48**	**91518.96**	0.69	86333.52	72.40	86881.42	93.99	89211.10	121.07
in207	**93129.25**	3.03	**93129.25**	2.18	89753.32	71.05	91033.51	100.90	92042.88	123.39
in208	**94904.68**	**0.46**	**94904.68**	0.48	85927.42	75.51	83667.76	101.29	87803.87	80.96
in209	**87268.97**	**7.16**	**87268.97**	17.45	84752.54	73.26	81966.65	96.42	85265.19	84.45
in210	**89962.40**	0.73	**89962.40**	0.61	86229.86	71.30	85079.98	97.78	87917.57	84.54

Table 6. Results on subset of REL_500_1000.

Instances	DHS-ACO		abcWDP		DCM		MA		HBHSA	
	f	*t*	*f*	*t*	*f*	*t*	*f*	*t*	*f*	*t*
in401	**77417.48**	**0.03**	**77417.48**	0.04	75438.49	26.59	72948.07	37.07	76035.94	40.63
in402	**76273.34**	**0.13**	**76273.34**	0.25	75146.65	24.51	71454.78	37.20	**76273.34**	40.70
in403	**74843.96**	0.02	**74843.96**	0.01	71309.10	25.70	**74843.96**	38.81	72465.39	41.04
in404	**78761.69**	0.06	**78761.69**	0.03	76877.34	26.42	78761.68	38.78	77091.37	38.37
in405	**75915.90**	0.22	**75915.90**	0.14	75104.28	28.18	72674.25	39.29	75684.28	40.38
in406	**72863.32**	**0.03**	**72863.32**	0.05	72055.10	28.31	71791.03	38.09	72203.10	37.46
in407	**76365.72**	0.13	**76365.72**	0.04	74443.52	28.45	73935.28	40.95	73650.63	36.00
in408	**77018.83**	**0.05**	**77018.83**	0.08	74766.64	27.55	72580.04	39.07	74747.72	38.66
in409	**73188.62**	0.03	**73188.62**	0.03	71965.11	25.62	68724.53	36.28	71924.64	40.47
in410	**73791.66**	0.06	**73791.66**	0.06	73092.48	25.49	71791.57	41.90	73726.39	40.22

Table 7. Results on subset of REL_1500_1000.

Instances	DHS-ACO		abcWDP		DCM		MA		HBHSA	
	f	*t*	*f*	*t*	*f*	*t*	*f*	*t*	*f*	*t*
in501	**88656.96**	**3.13**	**88656.96**	3.75	86353.64	149.28	79132.03	107.82	87341.22	196.18
in502	**86236.91**	1.33	**86236.91**	1.09	84207.64	128.05	80340.76	108.71	84896.64	204.67
in503	**87812.38**	12.80	**87812.38**	10.51	84547.97	137.57	83277.71	114.15	86313.22	197.64
in504	**85600.00**	**6.70**	**85600.00**	19.69	82742.72	139.61	81903.02	116.11	84604.71	202.92

Table 8. Results on subset of REL_1500_1500.

Instances	DHS-ACO		abcWDP		DCM		MA		HBHSA	
	f	t	f	t	f	t	f	t	f	t
in601	**108800.45**	1.78	**108800.45**	1.57	103273.33	154.69	99044.32	110.62	104402.60	191.21
in602	**105611.48**	4.44	**105611.48**	3.84	102390.49	144.28	98164.23	114.18	103152.40	202.68
in603	**105121.02**	**6.56**	**105121.02**	7.46	98794.90	137.20	94126.96	110.71	104928.00	187.35
in604	**107733.81**	**8.76**	**107733.81**	12.83	103522.86	138.49	103568.86	110.60	106694.10	197.08
in605	**109840.98**	**2.12**	**109840.98**	7.74	103600.76	143.48	102404.76	122.40	106322.10	188.35
in606	**107113.07**	**0.59**	**107113.07**	1.47	102906.98	141.04	104346.07	107.79	104499.70	195.60
in607	**113180.28**	2.63	**113180.28**	2.55	103297.49	141.11	105869.44	113.26	108241.00	197.03
in608	**105266.11**	**10.95**	**105266.11**	38.20	100547.10	139.90	95671.77	109.15	104428.30	197.22
in609	**109472.33**	2.91	**109472.33**	0.78	102506.90	139.33	98566.94	111.12	106122.20	198.18
in610	**113716.97**	**20.14**	**113716.97**	62.95	109516.88	138.17	102468.60	120.17	**113716.97**	198.16

Table 9. Results on basic benchmarks.

Class	DHS-ACO		abcWDP		DCM		MA		HBHSA	
	avg_f	*total_t*	*avg_f*	*total_t*	*avg_f*	*total_t*	*avg_f*	*total_t*	*avg_f*	*total_t*
REL_1000_500	**72028.45**	**25.42**	**72028.45**	102.55	67967.09	437.068	66622.55	1426.35	69183.89	589.25
REL_1000_1000	**88888.07**	**16.14**	**88888.07**	28.82	85105.16	732.68	84727.08	986.26	86778.28	943.88
REL_ 500_1000	**75644.05**	0.76	**75644.05**	**0.73**	74019.87	266.814	72950.52	387.44	74380.28	393.93
REL_1500_1000	**87076.56**	**23.96**	**87076.56**	35.04	84462.99	554.518	81163.38	446.79	85788.95	801.41
REL_1500_1500	**108585.65**	**60.88**	**108585.65**	139.39	103035.77	1417.688	100423.2	1130	106250.7	1952.86

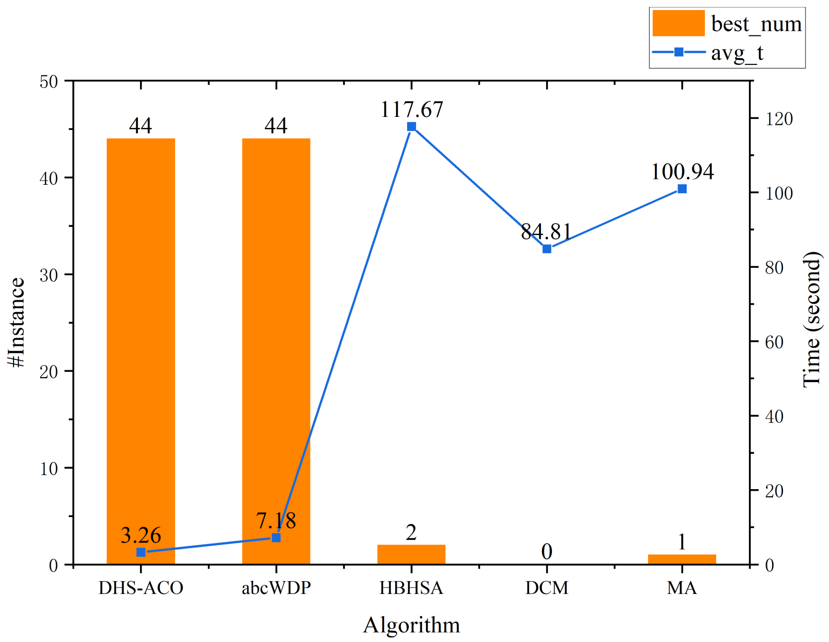

Figure 2. Results of *best_num* and *avg_t* on basic benchmarks.

Table 10. Results on extended benchmarks.

Instances	DHS-ACO		abcWDP	
	f	t	f	t
CATS_2000	**92095.11**	222.20	88243.66	306.92
CATS_2500	**108426.00**	382.62	103076.51	350.32
CATS_3000	**134062.11**	303.03	125171.43	195.10
CATS_3500	**154289.60**	306.95	143439.39	323.06
CATS_4000	**177537.83**	301.17	165046.76	297.31
CATS_4500	**203493.37**	360.22	186831.08	322.06
CATS_5000	**223222.27**	358.04	204797.15	257.33
CATS_5500	**242498.90**	314.37	223389.47	271.43
CATS_6000	**268460.02**	433.72	247635.80	310.24
CATS_6500	**286753.35**	424.86	262688.31	295.27
CATS_7000	**293800.27**	428.23	270759.39	402.83
CATS_7500	**332319.45**	508.10	304366.45	417.81
CATS_8000	**339733.32**	397.65	309655.66	218.48
Total (s)		4741.16		3968.16

5.5. Validation experiment

In order to verify the effectiveness of each key component of DHS-ACO, we compare DHS-ACO with its four variants, which are introduced as follows:

- DHS-ACOnoh: DHS-ACO algorithm without using the hash tabu strategy.
- DHS-ACOnofbs: DHS-ACO algorithm without using the deep scoring strategy.
- DHS-ACOnols: DHS-ACO algorithm without local search.
- DHS-ACOnoaco: DHS-ACO algorithm without ant colony framework. This means DHS-ACOnoaco only uses the local search to solve the benchmarks. When the hash tabu strategy decides to rebuild the solution, the solution will be initialized randomly rather than use the construction method of ACO.

Since the comparison experiment shows that DHS-ACO algorithm can solve all basic test cases stably, we select all extended cases in Table 2 to test the effectiveness of each key component. The comparisons between DHS-ACO and its variants are listed in Table 11 to Table 14.

Table 11 shows that DHS-ACOnoh algorithm gains lower f values than DHS-ACO on 11 instances, while obtains higher values on the remaining 2 instances. It can be seen that the intervention of the hash tabu strategy of the local search shows a negative effect, because the local search selects bids with randomness, which means the subsequent searches for those solutions marked by hash tabu strategy have little chance to find better solutions. However, in general, the hash tabu strategy makes DHS-ACO avoid repeated visiting efficiently and improve the local search significantly.

As can be seen from Table 12, DHS-ACO outperforms DHS-ACOnofbs on 10 out of 13 instances. One can see that the deep scoring strategy is counterproductive in a few test cases, because it focuses too much on the selection and maintenance of FB, and ignores the bids outside FB. Nevertheless, the deep scoring strategy improves the performance of DHS-ACO from an overall perspective.

According to Table 13, DHS-ACOnols obtains worse solutions on all instances without using the local search. Meanwhile, its convergence time is greatly reduced compared with DHS-ACO. It is clear that the proposed local search provides a powerful neighborhood search capability for DHS-ACO, which makes DHS-ACO effective in exploiting the solution space.

Table 14 indicates the effectiveness of the ant colony algorithm framework of DHS-ACO. Without the new solutions generated by the ant colony framework, DHS-ACOnoaco performs significantly worse than DHS-ACO on all test cases. It demonstrates the necessary of using the ant colony framework to provide high-quality solutions for DHS-ACO based on pheromones and heuristics.

Table 11. Results obtained by DHS-ACO and DHS-ACOnoh.

Instances	DHS-ACO		DHS-ACOnoh	
	f	*t*	*f*	*t*
CATS_2000	**92095.11**	222.20	92053.91	283.39
CATS_2500	**108426.00**	382.62	108262.94	315.93
CATS_3000	**134062.11**	303.03	133861.27	328.99
CATS_3500	**154289.60**	306.95	154234.50	341.62
CATS_4000	**177537.83**	301.17	177404.69	390.68
CATS_4500	203493.37	360.22	**203570.93**	397.36
CATS_5000	**223222.27**	358.04	222909.74	365.98
CATS_5500	**242498.90**	314.37	242434.70	392.56
CATS_6000	**268460.02**	433.72	268433.42	418.30
CATS_6500	**286753.35**	424.86	286460.81	399.11
CATS_7000	293800.27	428.23	**294079.34**	419.63
CATS_7500	**332319.45**	508.10	331237.09	446.86
CATS_8000	**339733.32**	397.65	339644.22	339.03

Table 12. Results obtained by DHS-ACO and DHS-ACOnofbs.

Instances	DHS-ACO		DHS-ACOnofbs	
	f	*t*	*f*	*t*
CATS_2000	92095.11	222.20	**92103.01**	305.01
CATS_2500	108426.00	382.62	**109000.30**	202.89
CATS_3000	**134062.11**	303.03	133422.57	375.12
CATS_3500	**154289.60**	306.95	154051.85	307.68
CATS_4000	**177537.83**	301.17	176574.29	213.85
CATS_4500	203493.37	360.22	**203507.10**	379.32
CATS_5000	**223222.27**	358.04	222270.79	414.67
CATS_5500	**242498.90**	314.37	241873.75	375.88
CATS_6000	**268460.02**	433.72	267820.75	463.77
CATS_6500	**286753.35**	424.86	286299.25	424.79
CATS_7000	**293800.27**	428.23	293537.66	437.80
CATS_7500	**332319.45**	508.10	331171.41	462.54
CATS_8000	**339733.32**	397.65	338324.77	461.95

Table 13. Results obtained by DHS-ACO and DHS-ACOnols.

Instances	DHS-ACO		DHS-ACOnols	
	f	*t*	f	*t*
CATS_2000	**92095.11**	222.20	85292.66	2.65
CATS_2500	**108426.00**	382.62	100334.73	3.14
CATS_3000	**134062.11**	303.03	122329.02	5.69
CATS_3500	**154289.60**	306.95	141815.45	7.67
CATS_4000	**177537.83**	301.17	162805.02	10.27
CATS_4500	**203493.37**	360.22	188135.22	13.94
CATS_5000	**223222.27**	358.04	205140.48	19.58
CATS_5500	**242498.90**	314.37	223106.47	25.55
CATS_6000	**268460.02**	433.72	249082.54	30.69
CATS_6500	**286753.35**	424.86	266662.91	35.89
CATS_7000	**293800.27**	428.23	270262.84	39.84
CATS_7500	**332319.45**	508.10	305819.14	51.85
CATS_8000	**339733.32**	397.65	312315.06	56.36

Table 14. Results obtained by DHS-ACO and DHS-ACOnoaco.

Instances	DHS-ACO		DHS-ACOnoaco	
	f	*t*	*f*	*t*
CATS_2000	**92095.11**	222.20	87859.02	239.43
CATS_2500	**108426.00**	382.62	102506.50	171.21
CATS_3000	**134062.11**	303.03	124633.43	199.47
CATS_3500	**154289.60**	306.95	143283.79	211.61
CATS_4000	**177537.83**	301.17	165032.62	147.11
CATS_4500	**203493.37**	360.22	186365.41	240.25
CATS_5000	**223222.27**	358.04	204715.67	180.20
CATS_5500	**242498.90**	314.37	223167.11	141.10
CATS_6000	**268460.02**	433.72	246538.17	251.53
CATS_6500	**286753.35**	424.86	262135.68	250.56
CATS_7000	**293800.27**	428.23	270097.45	222.24
CATS_7500	**332319.45**	508.10	303594.47	309.95
CATS_8000	**339733.32**	397.65	309485.41	245.40

6. Conclusions

In this paper, we propose a new swarm algorithm called DHS-ACO, which can effectively deal with the WDP on a wide instances. Based on the ant colony framework, the algorithm combines an effective local search with a hash tabu method and a deep scoring strategy. Extensive computational evaluations of the algorithm on two set of benchmarks demonstrated its competitiveness compared to the state-of-the-art methods. In particular, the algorithm can find the same best solutions on all basic benchmarks with less time reported in the literature and the best solutions on all extended benchmarks.

Furthermore, owing to the random nature of the swarm intelligent algorithms, DHS-ACO need more time than its competitor abcWDP on large-scale benchmarks, although it can give better solutions. It would be interesting to reduce the consumption time on the large-scale benchmarks in the future.

Acknowledgments

This work is supported by the Fundamental Research Funds for the Central Universities 2412018QD022, NSFC (under Grant No.61976050, 61972384) and Jilin Provincial Science and Technology Department under Grant No. 20190302109GX.

Conflict of interest

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

1. S. J. Rassenti, V. L. Smith, R. L. Bulfin, A combinatorial auction mechanism for airport time slot allocation, *Bell J. Econ.*, **13** (1982), 402–417. https://doi.org/10.2307/3003463
2. K. Xu, Y. Zhang, X. Shi, H. Wang, Y. Wang, M. Shen, Online combinatorial double auction for mobile cloud computing markets, in *IEEE 33rd International Performance Computing and Communications Conference*, (2014), 1–8. https://doi.org/10.1109/PCCC.2014.7017103
3. T. G. Chetan, M. Jenamani, S. P. Sarmah, Two-stage multi-attribute auction mechanism for price discovery and winner determination, *Trans. Eng. Manage.*, **66** (2019), 112–126. https://doi.org/10.1109/TEM.2018.2810510
4. S. Wang, S. Qu, M. Goh, M. Wahab, H. Zhou, Integrated multi-stage decision-making for winner determination problem in online multi-attribute reverse auctions under uncertainty, *Int. J. Fuzzy Syst.*, **22** (2019), 2354–2372. https://doi.org/10.1007/s40815-019-00757-0
5. W. Fontanini, P. A. Ferreira, A game-theoretic approach for the web services scheduling problem, *Expert Syst. Appl.*, **41** (2014), 4743–4751. https://doi.org/10.1016/j.eswa.2014.02.016
6. D. H. Kim, S. A. Kazmi, A. Ndikumana, A. Manzoor, W. Saad, C. S. Hong, et al., Distributed radio slice allocation in wireless network virtualization: matching theory meets auctions, *IEEE Access*, **8** (2020), 494–732. https://doi.org/10.1109/ACCESS.2020.2987753
7. A. K. Ray, M. Jenamani, P. K. Mohapatra, Supplier behavior modeling and winner determination using parallel mdp, *Expert Syst. Appl.*, **38** (2011), 4689–4697. https://doi.org/10.1016/j.eswa.2010.08.044
8. S. De Vries, R. V. Vohra, Combinatorial auctions: a survey, *INFORMS J. Comput.*, **15** (2003), 284–309. https://doi.org/10.1287/ijoc.15.3.284.16077
9. M. Rekik, S. Mellouli, Reputation-based winner determination problem for combinatorial transportation procurement auctions, *J. Oper. Res. Soc.*, **63** (2012), 1400–1409. https://doi.org/10.1057/jors.2011.108

10. W. Zhong, K. Xie, Y. Liu, C. Yang, S. Xie, Multi-resource allocation of shared energy storage: a distributed combinatorial auction approach, *IEEE Trans. Smart Grid*, **11** (2020), 4105–4115. https://doi.org/10.1109/TSG.2020.2986468

11. M. H. Rothkopf, A. Pekeč, R. M. Harstad, Computationally manageable combinational auctions, *Manag. Sci.*, **44** (1998), 1131–1147. https://doi.org/10.1287/mnsc.44.8.1131

12. X. Li, S. Ma, Multi-objective memetic search algorithm for multi-objective permutation flow shop scheduling problem, *IEEE Access*, **4** (2016), 2154–2165. https://doi.org/10.1109/ACCESS.2016.2565622

13. Z. Lu, Y. Zhou, J. K. Hao, A hybrid evolutionary algorithm for the clique partitioning problem, *IEEE Trans. Cybernetics*, (2021).

14. M. Aïder, O. Gacem, M. Hifi, A hybrid population-based algorithm for the bi-objective quadratic multiple knapsack problem, *Expert Syst. Appl.*, **191** (2022), 116238. https://doi.org/10.1016/j.eswa.2021.116238

15. Q. Zhou, J. K. Hao, Q. Wu, A hybrid evolutionary search for the generalized quadratic multiple knapsack problem, *Eur. J. Oper. Res.*, **296** (2022), 88–803. https://doi.org/10.1016/j.ejor.2021.04.001

16. X. Li, X. Zhang, M. Yin, J. Wang, A genetic algorithm for the distributed assembly permutation flowshop scheduling problem, in *2015 IEEE Congress on Evolutionary Computation (CEC)*, (2015), 3096–3101. https://doi.org/10.1109/CEC.2015.7257275

17. Y. Zhou, X. Liu, S. Hu, Y. Wang, M. Yin, Combining max-min ant system with effective local search for solving the maximum set k-covering problem, *Knowl. Based Syst.*, **239** (2021), 108000. https://doi.org/10.1016/j.knosys.2021.108000

18. Ş. Öztürk, R. Ahmad, N. Akhtar, Variants of artificial bee colony algorithm and its applications in medical image processing, *Appl. Soft Comput.*, **97** (2020), 106799. https://doi.org/10.1016/j.asoc.2020.106799

19. M. Dorigo, L. M. Gambardella, Ant colony system: a cooperative learning approach to the traveling salesman problem, *IEEE Trans. Evol. Comput.*, **1** (1997), 53–66. https://doi.org/10.1109/4235.585892

20. X. Zhang, X. Li, J. Wang, Local search algorithm with path relinking for single batch-processing machine scheduling problem, *Neural Comput. Appl.*, **28** (2017), 313–326. https://doi.org/10.1007/s00521-016-2339-z

21. M. Li, J. K. Hao, Q. Wu, Learning-driven feasible and infeasible tabu search for airport gate assignment, *Eur. J. Oper. Res.*, **2021** (2021). https://doi.org/10.1016/j.ejor.2021.12.019

22. Z. Lu, J. K. Hao, U. Benlic, D. Lesaint, Iterated multilevel simulated annealing for large-scale graph conductance minimization, *Inform. Sci.*, **572** (2021), 182–199. https://doi.org/10.1016/j.ins.2021.04.102

23. F. Glover, Tabu search-part i, *ORSA J. Comput.*, **1** (1989), 190–206. https://doi.org/10.1287/ijoc.1.3.190

24. Y. Zhou, J. Li, Y. Liu, S. Lv, Y. Lai, J. Wang, Improved memetic algorithm for solving the minimum weight vertex independent dominating set, *Mathematics*, **8** (2020), 1155. https://doi.org/10.3390/math8071155
25. P. V. Silvestrin, M. Ritt, An iterated tabu search for the multi-compartment vehicle routing problem, *Comput. & Oper. Res.*, **81** (2017), 192–202. https://doi.org/10.1016/j.cor.2016.12.023
26. L. Xing, Y. Liu, H. Li, C. C. Wu, W. C. Lin, X. Chen, A novel tabu search algorithm for multi-agv routing problem, *Mathematics*, **8** (2020), 279. https://doi.org/10.3390/math8020279
27. B. Vangerven, D. R. Goossens, F. C. Spieksma, Winner determination in geometrical combinatorial auctions, *Eur. J. Oper. Res.*, **258** (2017), 254–263. https://doi.org/10.1016/j.ejor.2016.08.037
28. M. Kaleta, Network winner determination problem, *Arch. Control Sci.*, **28** (2018). https://doi.org/10.24425/119077
29. N. Remli, A. Amrouss, I. El Hallaoui, M. Rekik, A robust optimization approach for the winner determination problem with uncertainty on shipment volumes and carriers' capacity, *Trans. Res. Part B: Meth.*, **123** (2019), 127–148. https://doi.org/10.1016/j.trb.2019.03.017
30. X. Qian, S. C. Fang, M. Huang, X. Wang, Winner determination of loss-averse buyers with incomplete information in multiattribute reverse auctions for clean energy device procurement, *Energy*, **177** (2019), 276–292. https://doi.org/10.1016/j.energy.2019.04.072
31. X. Qian, F. T. Chan, M. Yin, Q. Zhang, M. Huang, X. Fu, A two-stage stochastic winner determination model integrating a hybrid mitigation strategy for transportation service procurement auctions, *Comput. Ind. Eng.*, **149** (2020), 106703. https://doi.org/10.1016/j.cie.2020.106703
32. C. W. Lee, W. P. Wong, J. Ignatius, A. Rahman, M. L. Tseng, Winner determination problem in multiple automated guided vehicle considering cost and flexibility, *Comput. Ind. Eng.*, **142** (2020), 106337. https://doi.org/10.1016/j.cie.2020.106337
33. Y. Fujishima, K. Leyton-Brown, Y. Shoham, Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches, in *IJCAI*, **99** (1999), 548–553.
34. N. Nisan, Bidding and allocation in combinatorial auctions, in *Proceedings of the 2nd ACM Conference on Electronic Commerce*, (2000), 1–12. https://doi.org/10.1145/352871.352872
35. K. Leyton-Brown, Y. Shoham, M. Tennenholtz, An algorithm for multi-unit combinatorial auctions, in *Aaai/iaai*, (2000), 56–61.
36. T. Sandholm, S. Suri, Bob: Improved winner determination in combinatorial auctions and generalizations, *Artif. Intell.*, **145** (2003), 33–58. https://doi.org/10.1016/S0004-3702(03)00015-8
37. O. Günlük, L. Ladányi, S. De Vries, A branch-and-price algorithm and new test problems for spectrum auctions, *Manag. Sci.*, **51** (2005), 391–406. https://doi.org/10.1287/mnsc.1040.0332
38. L. F. Escudero, M. Landete, A. Marín, A branch-and-cut algorithm for the winner determination problem, *Decis. Support Syst.*, **46** (2009), 649–659. https://doi.org/10.1016/j.dss.2008.10.009
39. H. H. Hoos, C. Boutilier, Solving combinatorial auctions using stochastic local search, in *Aaai/iaai*, (2000), 22–29.
40. Y. Guo, A. Lim, B. Rodrigues, Y. Zhu, Heuristics for a bidding problem, *Comput. Oper. Res.*, **33** (2006), 2179–2188.

41. D. Boughaci, B. Benhamou, H. Drias, Stochastic local search for the optimal winner determination problem in combinatorial auctions, in *International Conference on Principles and Practice of Constraint Programming*, Springer, (2008), 593–597.
42. N. Wang, D. Wang, Model and algorithm of winner determination problem in multi-item e-procurement with variable quantities, in *The 26th Chinese Control and Decision Conference (2014 CCDC)*, (2014), 5364–5367. https://doi.org/10.1109/CCDC.2014.6852222
43. M. B. Dowlatshahi, V. Derhami, Winner determination in combinatorial auctions using hybrid ant colony optimization and multi-neighborhood local search, *J. AI Data Min.*, **5** (2017), 169–181.
44. D. Boughaci, B. Benhamou, H. Drias, A memetic algorithm for the optimal winner determination problem, *Soft Comput.*, **13** (2009), 905. https://doi.org/10.1007/s00500-008-0355-3
45. H. Zhang, S. Cai, C. Luo, M. Yin, An efficient local search algorithm for the winner determination problem, *J. Heuristics*, **23** (2017), 367–396. https://doi.org/10.1007/s10732-017-9344-y
46. G. Lin, W. Zhu, M. M. Ali, An effective discrete dynamic convexized method for solving the winner determination problem, *J. Comb. Optim.*, **32** (2016), 563–593. https://doi.org/10.1007/s10878-015-9883-9
47. G. Lin, Z. Li, A hybrid binary harmony search algorithm for solving the winner determination problem, *Int. J. Innovative Comput. Appl.*, **10** (2019), 59–68. https://doi.org/10.1504/IJICA.2019.100547
48. H. C. Lau, Y. G. Goh, An intelligent brokering system to support multi-agent web-based 4/sup th/-party logistics, in *14th IEEE International Conference on Tools with Artificial Intelligence*, (2002), 154–161.
49. K. Leyton-Brown, M. Pearson, Y. Shoham, Towards a universal test suite for combinatorial auction algorithms, in *Proceedings of the 2nd ACM Conference on Electronic Commerce*, (2000), 66–76. https://doi.org/10.1145/352871.352879
50. M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle, The irace package: Iterated racing for automatic algorithm configuration, *Oper. Res. Perspect.*, **3** (2016), 43–58. https://doi.org/10.1016/j.orp.2016.09.002