Hindawi
Mathematical Problems in Engineering
Volume 2020, Article ID 3050714, 11 pages
https://doi.org/10.1155/2020/3050714

Research Article

Solving the Set Packing Problem via a Maximum Weighted Independent Set Heuristic

Ruizhi Li,[1,2] Yupan Wang,[3] Shuli Hu,[3] Jianhua Jiang,[1] Dantong Ouyang,[2] and Minghao Yin[3,4]

[1]*School of Management Science and Information Engineering, Jilin University of Finance and Economics, Changchun 130117, China*
[2]*School of Computer Science and Technology, Jilin University, Changchun 130012, China*
[3]*School of Computer Science and Information Technology, Northeast Normal University, Changchun 130024, China*
[4]*Key Laboratory of Applied Statistics of MOE, Northeast Normal University, Changchun 130117, China*

Correspondence should be addressed to Dantong Ouyang; ouyd@jlu.edu.cn and Minghao Yin; ymh@nenu.edu.cn

Received 13 July 2020; Revised 15 October 2020; Accepted 18 November 2020; Published 16 December 2020

Academic Editor: Thomas Hanne

The set packing problem (SPP) is a significant NP-hard combinatorial optimization problem with extensive applications. In this paper, we encode the set packing problem as the maximum weighted independent set (MWIS) problem and solve the encoded problem with an efficient algorithm designed to the MWIS problem. We compare the independent set-based method with the state-of-the-art algorithms for the set packing problem on the 64 standard benchmark instances. The experimental results show that the independent set-based method is superior to the existing algorithms in terms of the quality of the solutions and running time obtained the solutions.

1. Introduction

The set packing problem is a classical combinatorial optimization problem and it has been studied extensively by researchers in recent years. In the set packing problem, there are n objects and a set of exclusive constraints between some objects of OB labelled as $O_1, \ldots, O_m$. Each object $j \in OB$ is associated with a positive weight c_j. The aim of set packing problem is to find out a packing that maximizes the total weight of objects such that any constraint should not be violated. The problem is widely used in various fields such as routing and scheduling trains at intersections in railway operations [1], selecting winning bids in combinatorial auctions [2], surgical operations scheduling [3], and packets scheduling and transmission in communication networks [4] among many others.

The set packing problem is an NP-hard problem [5]. The solving algorithms for this problem can be categorized into two types of exact and inexact ones. In [6–8], new facets were identified for the polyhedron of the problem, which strengthen the solution of the relaxed problem. Rossi and Smriglio used a branch and cut algorithm to solve the set packing problems [9]. Kwon et al. proposed an approach for ex-postevaluation of approximate solutions obtained by a well-known simple greedy method for set packing [10]. Kolokolov and Zaozerskaya found the polynomial upper bounds on average iterations number for L-class enumeration algorithm and the first Gomory cutting plane algorithm [11]. Landete et al. presented an alternative formulation for the set packing problem in a higher dimension and the addition of a new family of binary variables allowed the authors to find new valid inequalities, some of which were shown to be facets of the polytope in the higher dimension [12]. However, the computational time required for this exact approach increases exponentially with the size of the problem in general. The exact approach can only obtain optimal solutions for relatively small-scale instances. So as to solve larger-scale instances, heuristic algorithms [13–17], which indeed play an important role in obtaining high-quality solutions to combinatorial optimization problems in a reasonable time [18–22], have been designed. For set packing problem, Rönnqvist proposed a combination of Lagrangian relaxation and the subgradient method to solve the cutting stock problem, which is an application of the set packing problem [23]. Chandra and Halldórsson proposed a combination of greedy algorithms and local search methods to tackle the special cases of the set packing problem [24]. Lau and Goh presented a greedy algorithm to solving the set packing problem [25]. In the literature [26], a greedy randomized adaptive search procedure (GRASP) was proposed, and railway problem instances and random instances were tested to measure the effectiveness of GRASP [26]. In literature [27], Gandibleux et al. proposed an ant colony optimization (ACO) method, and the random instances were used to evaluate the ACO algorithm. Guo et at. presented the simulated annealing heuristic with three local moves to solving the bidding problem which can be modelled as the set packing problem [28]. The set packing problem was modelled as the unconstrained quadratic binary program and Tabu search was proposed to solve it [29]. Later, two improved versions of ACO were proposed in [30, 31]. In [32], an approximation algorithm based on local search methods was proposed. EA/G [33] was a recently proposed evolutionary algorithm and was applied on random instances. Chaurasia et al. presented an evolutionary algorithm-based hyperheuristic framework for solving the set packing problem [34]. Chaurasia and Kim proposed an evolutionary algorithm-based hyperheuristic framework that incorporates dynamic selection of parameters [35]. In [36], a decomposition technique based on constraint partitioning was proposed for exploiting the semiblock-angular structures of set packing problem and solving the original problem through solving the subproblems of the obtained structure.

Although some heuristic search algorithms have been used to solve the set packing problem, their solving quality still needs to be improved. In particular, the average solution obtained by the existing algorithms differs a lot from the optimal solution. In other words, the stability of the solution obtained by the existing algorithms is not very good and has a lot of randomness. Therefore, a new idea is proposed to solve this problem, which is to transform the set packing problem to the weighted independent set problem. As a result, any solving method proposed for minimum weighted independent set can be used to solve SPP via its independent formulation. The idea of encoding a problem into an equivalent problem to solve is appealing because of two reasons: one is that we can solve the set packing problem without designing dedicated set packing solving algorithms. The other is that we can make full use of the minimum weighted independent solving algorithms to better solve the set packing problem. Furthermore, we can even use different minimum weighted independent set algorithms to enlarge the scale of solvable instances of the set packing problem. In our paper, we solve the set packing problem by using the efficient algorithm proposed previously for minimum weighted independent set problem. To our knowledge, the minimum weighted independent set-based methods for the set packing problem have not been used before.

The rest of this article is structured as follows. In Section 2, we introduce the method of encoding set packing problem as minimum weighted independent problem. Then, we briefly review the algorithm DLSWCC (diversion local search based on weighted configuration checking) for set packing problem in Section 3. The experimental results and the analyses of the experimental results on standard benchmarks are presented in Section 4. The conclusions and perspectives for future work are shown in Section 5.

2. The Encoding Method

In this section, we introduce the concepts used in this article and the encoding method. According to the definitions in literatures [26, 27], the set packing problem can be described as follows. Given an object set $OB = \{1, \ldots, n\}$ and each object $i \in OB$ is associated with a positive weight c_i, we use O_j to represent a set of exclusive constraints between some objects of OB, where $j \in J = \{1, \ldots, m\}$. A packing PA is a subset of OB such that $|O_j \cap PA| \le 1$, $\forall j \in J$, i.e., at most one object of O_j can be in PA.

The aim of the set packing problem is to obtain a packing that maximizes the total weight of the contained objects without violating any constraint. Formally, the set packing problem can be formulated as follows:

$$\text{maximize } z = \sum_{i \in OB} c_i x_i, \tag{1}$$

$$\text{subject to } \sum_{i \in OB} o_{i,j} x_i \le 1, \quad \forall i \in OB, \forall j \in OB, \tag{2}$$

$$x_i \in \{0, 1\}, \quad \forall i \in OB, \tag{3}$$

$$o_{i,j} \in \{0, 1\}, \quad \forall i \in OB, \forall j \in OB. \tag{4}$$

where x_i is a binary variable, indicating whether object i is in PA, $x_i = 1$ means in PA, otherwise, $x_i = 0$ means not in PA, c_i is the weight of object i, and $o_{i,j}$ is also a binary variable, indicating whether object i belongs to exclusive constraint set O_j, $o_{i,j} = 1$ means belonging to set O_j, $o_{i,j} = 0$ means not belonging to set O_j. Formula (1) is the objective function of maximizing the total weight of objects belonging to PA. Constraint (2) establishes that at most one object of O_j can be in PA. Equations (3) and (4) are integer constraints.

The set packing problem can be conveniently encoded as the maximum weighted independent set problem. To see this, we first review some basic symbols associated with the MWIS problem. Given an undirected graph $G = (Vt, Eg, w)$, $Vt = \{1, \ldots, n\}$ is the vertex set, $Eg \subset Vt \times Vt$ is the edge set, w is the vertex weighting function, and each vertex $i \in Vt$ is assigned a positive integer weight w_i. An independent set IS of G is a subset of Vt such that there is no pair of vertices in IS linked by an edge in Eg, i.e., $\forall u, v \in IS$, $\{u, v\} \notin Eg$. The weight of an independent set IS of G is the sum of all contained vertices' weights, i.e., $w(IS) = \sum_{i \in IS} w_i$. Then, the maximum weighted independent set problem is to find the independent set with the maximum sum of weights of the contained vertices.

For a set packing problem instance, an object set $OB = \{1, \ldots, n\}$, and an exclusive constraint O_j, $j \in J = \{1, \ldots, m\}$, each object $u \in OB$ is associated with a positive weight c_u. We give a maximum weighted independent set instance (conflict graph) $G = (Vt, Eg, w)$ as follows:

(i) For an object $u \in OB$, define a vertex $u \in Vt$, whose weight w_u is equal to c_u. That is, $Vt = \{1, \ldots, n\}$, $\forall u \in V$, $w_u = c_u$.

(ii) For the exclusive constraint, define the edge matrix Eg by

$$e_{uv} = \begin{cases} 1, & \text{if } u \in O_j \text{ and } v \in O_j, u, v \in \{1, \ldots, n\}, j \in \{1, \ldots, m\}, \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

It is easy to see an edge e_{uv} will link two vertices u and v if the objects u and v are in the same exclusive constraint, which indicates that the two objects are in conflict and cannot be accepted at the same time. From the edge matrices, we can easily check the objects that are included in the same exclusive constraint.

According to the above transformation, it is not difficult to find that a maximum weighted independent set $IS = \{v_1, \ldots, v_r\}$ of the conflict graph $G = (Vt, Eg, w)$ corresponding to a feasible set packing $PA = \{1, \ldots, r\}$ of objects with maximum of the total weight of objects it contains without violating any constraint. So, any solving algorithm for the minimum weighted independent set problem can be applied to tackle the set packing problem.

To further illustrate the encoding, we give an example, i.e., a set packing problem instance with 6 objects and 4 constraints in Figure 1(a). The correspondent conflict graph $G = (Vt, Eg, w)$ with regard to the set packing problem instance is described in Figure 1(b) where each object is represented by a vertex whose weight is equal to that of the object. An edge will link two vertices if they are included in the same constraint. It is distinct that the optimal solution to the maximum weighted independent set problem defined by the conflict graph is given by the vertex set $\{v_1, v_3, v_6\}$ which represents the set of objects $\{1, 3, 6\}$ with a maximum weight of 29.

3. Diversion Local Search Based on Weighted Configuration Checking for SPP

Given a set packing problem instance, we can encode this instance as a maximum weighted independent set problem instance according to the previous section. So, any algorithm to solve the maximum weighted independent set can be used to solve the set packing problem. As far as we know, the maximum weighted independent set problem has two equivalent problems, namely, minimum weighted vertex cover (MWVC) problem and maximum weighted clique (MWC) problem. Methods for solving MWVC problem can be directly applied to tackle the MWIS problem. In this paper, an efficient local search algorithm called "Diversion Local Search based on Weighted Configuration Checking (DLSWCC)" is used to solve the MWIS problem [37]. The algorithm has high efficiency in solving minimum weighted vertex cover and set packing problem. Let us briefly review the key factors of the DLSWCC algorithm. For a comprehensive description, the readers can refer to the literature [37].

3.1. Dynamic Scoring Strategy.

For a candidate solution, the quality of the candidate solution can be improved by selecting appropriate vertices to add to or remove from the candidate solution, thus improving the performance of the local search algorithm. We present a dynamic scoring strategy to evaluate the benefit when the vertex is added to or deleted from the candidate solution. The dynamic edge weighting mechanism is used in dynamic scoring strategy. The definition of dynamic edge weight is given below.

Definition 1 (dynamic edge weight). Given an undirected graph G (Vt, Eg, w), each edge $e \in Eg$ is assigned a weight denoted by *dynmc_w* (e), and the weight is dynamically updated in the local search.

Specifically, we abide by the following two rules to update edge weights.

W_Rule1: the *dynmc_w* (e) of each edge $e \in Eg$ is initialized as 1

W_Rule2: the *dynmc_w* (e) will be increased by 1 if edge e is not covered by the candidate solution at the end of each loop

On the basis of Definition 1, assuming that vertex subset $C \subseteq Vt$ is a candidate solution, the Boolean function cover (e, C) is used to indicate whether edge $e \in Eg$ is covered by candidate solution C, i.e., whether at least one of e's endpoints is a member of C. The quality of the candidate solution C is measured by cost (C), which is defined by the following formula:

$$\cos t\,(C) = \sum_{\text{cover}\,(e,C)=false} dynmc\_w\,(e). \tag{6}$$

From Formula (6), we can see that cost (C) represents the weight sum of the edge uncovered by candidate solution C. The candidate solution C is feasible if cost (C) is equal to 0.

The score of vertices is crucial to choose which vertex to add to or remove from the candidate solution. In algorithm DLSWCC, the authors use score (v) to define the score of vertex v, as shown in the following formula:

$$\text{score}\,(v) = \frac{\cos t\,(C) - \cos t\,(C')}{w_v}, \tag{7}$$

where C is the current candidate solution, if v belongs to C, then C' is the candidate solution after removing vertex v from C; otherwise, C' is the candidate solution after adding vertex v to C.

3.2. Weighted Configuration Checking Strategy.

The cycling problem is revisiting a scenario that has just been visited during the local search phase. This problem will make the algorithm fall into the local optimum, cause the waste of

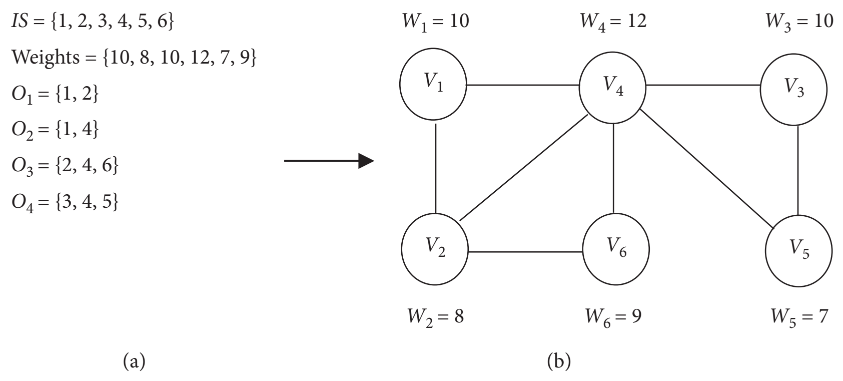

Figure 1: The original set packing problem instance (a) and the associated maximum weighted independent set problem instance (b).

time, and reduce the performance of the algorithm. Many scholars have been working on how to avoid the cycling problem. In literature [38], Cai et al. proposed the configuration checking strategy, which can consider the environmental information to avoid the cycling problem. Up to now, CC strategy has been successfully used to tackle many combinatorial optimization problems, i.e., the minimum vertex cover, SAT, MaxSAT, and set cover problem [39–41].

However, the direct application of CC strategy to MWVC problem will limit some promising vertices to be added into the candidate solution, thus misleading the search. That is, the original CC strategy is more restrictive. In our algorithm, we will use the deformation strategy of CC strategy, namely, weighted configuration checking (WCC) strategy. The concept of weighted configuration is given below.

Definition 2. (weighted configuration). Given an undirected graph $G = (Vt, Eg)$, each edge is associated with a weight, and C is the candidate solution. The weighted configuration of vertex v is defined as the states of all v's neighbours and the weights of the associated edges of all v's neighbours.

In order to implement the weighted configuration checking (WCC) strategy, we use an array *wcnfg* to record whether the weighted configuration of each vertex has changed since last leaving C. Each element of the array is a binary variable. For a vertex v, *wcnfg* $[v] = 1$ indicates that the weighted configuration of vertex v has changed and *wcnfg* $[v] = 0$ on the contrary. We update the *wcnfg* array according to the following four rules:

(i) WCC_Rule1: in the initialization phase, the *wcnfg* value of each vertex v is assigned to 1

(ii) WCC_Rule2: if vertex v is removed from C, then the *wcnfg* value of v is assigned to 0 and the *wcnfg* value of v's each neighbour is assigned to 1

(iii) WCC_Rule3: if vertex v is added into C, then the *wcnfg* value of v's each neighbour is assigned to 1

(iv) WCC_Rule4: if edge e's weight *dynmc_w* $[e]$ is updated, then the *wcnfg* values of the two vertices u and v linked by edge e are assigned to 1

3.3. Vertex Selection Strategy. In this subsection, we introduce the vertex selection strategy, which combines the dynamic scoring strategy with the weighted configuration checking strategy. Before we introduce this strategy, let us introduce the age concept that we will be using. The age of a vertex is the number of iterations after the vertex's state has changed.

In the local search phase, we use the following two rules to select suitable vertices to add to or remove from the candidate solution.

(i) Rmv_Rule: the vertex with the highest score is selected from the candidate solution, or the vertex with the greatest age is selected if there are multiple vertices to choose from. Then, *wcnfg* values of this selected vertex and its neighbours are modified according to WCC_Rule2.

(ii) Add_Rule: the vertex with the highest score and *wcnfg* value of 1 is selected from the vertices of the noncandidate solution. If there are multiple optional vertices, the vertex with the greatest age is selected. Then, *wcnfg* values of its neighbours are modified according to WCC_Rule3.

3.4. DLSWCC Algorithm. In this subsection, we review the main idea of DLSWCC algorithm, and the corresponding pseudocode is shown in Algorithm 1. First, we construct the initial solution C by the greedy method. Then, a perturbing approach is applied on the initial solution C to improve its quality. We use $w(C) = \sum_{i \in C} w_i$ to represent the objective value of the candidate solution C. We use UB to record the objective value of the global optimal solution and initialize UB to $w(C)$. It is obvious that if a better solution exists, the objective value should be less than UB. In DLSWCC algorithm, once the initial candidate solution has been built, we will remove some vertices from the candidate solution until the candidate solution becomes infeasible and the objective value is less than UB. Then, we exchange the vertices in C and the vertices in $Vt \backslash C$ according to Rmv_Rule and Add_Rule until C is a feasible solution. At this stage, if a better solution is found, the value of UB needs to be updated. At the end of each loop, the algorithm checks if each edge is covered by the current solution, and if not, the algorithm

```
(1) Initialize $wcnfg$ array according to **WCC_Rule1**;
(2) initialize the $dynmc\_w$ of each edge assigned as 1;
(3) initialize the score of each vertex assigned as the degree of the vertex;
(4) initialize the candidate solution $C$ greedily;
(5) $UB = w(C)$;
(6) $C^* \longleftarrow C$;
(7) iter $\longleftarrow$ 0;
(8) while stop criterion is not satisfied do
(9)   while $C$ covers all edges, then
(10)      $\mathrm{UB} = w(C)$;
(11)      $C^* \longleftarrow C$;
(12)      $v \longleftarrow x$ with the greatest score in $C$, breaking ties in favor of the oldest one;
(13)      $C \longleftarrow C \backslash \{v\}$;
(14)      update $wcnfg$ array according to **WCC_Rule 2**;
(15)  end while
(16)    $v \longleftarrow x$ with the greatest score in $C$ and $v$ is not in tabu_list, breaking ties in favor of the oldest one;
(17)    $C \longleftarrow C \backslash \{v\}$;
(18)    update $wcnfg$ array according to **WCC_Rule 2**;
(19)    clear $tabu\_list$;
(20)    while $C$ uncovers some edges do
(21)      $v \longleftarrow x$ with the greatest score not in $C$ and $wcnfg[x] == 1$, breaking ties in favor of the oldest one;
(22)      if $w(C) + w(v) \geq \mathrm{UB}$ then break;
(23)      $C \longleftarrow C \cup \{v\}$;
(24)      update $wcnfg$ array according to **WCC_Rule3**;
(25)      $dynmc\_w[e] \longleftarrow dynmc\_w[e] + 1$, for each uncovered edge by $C$;
(26)      update $wcnfg$ array according to **WCC_Rule4**;
(27)      add $v$ into tabu_list;
(28)    end while
(29)    iter $\longleftarrow$ iter + 1;
(30) end while
(31) return $C*$;
```

Algorithm 1: DLSWCC ( ).

adds the weight *dynmc_w* of the uncovered edge by 1, thus giving the "hard to cover" edges a better chance to be covered by the new candidate solution in the future iterations and making the algorithm jump out of local optimum effectively.

4. Computational Results

In this section, we will report a large number of experimental results through using the introduced DLSWCC algorithm to solve the set packing problem as a minimum weighted independent set on a large number of set packing problem standard benchmarks. Further, DLSWCC is compared with several state-of-the-art algorithms proposed in the literature. Finally, we test the effectiveness of the dynamic scoring strategy and the weighted configuration checking strategy.

4.1. Reference Algorithms and Experimental Protocol. We compare DLSWCC with the current best solving algorithms, i.e., CPLEX, GRASP approach [26], ACO approach [27], and EA/G approach [33]. In this study, our DLSWCC algorithm is implemented in C and executed on a computer with Intel (R) Xeon (R) CPU E7-4830 with 2.13 GHz. The system that is used to execute ACO and GRASP approaches [27] is Pentium III at 800 MHz. EA/G approach [33] is implemented in C and executed on a Core 2 Duo system with 2 GB RAM running under Fedora 12 at 3.0 GHz. For each instance, our algorithm DLSWCC is executed, where the cutoff condition for each execution is to reach a given cutoff time 3600 (s) or max iteration 1000000. Like EA/G [33], GRASP [26], and ACO [27] approaches, DLSWCC is run 16 times independently on each instance.

The railway problem instances and random instances are two main types of standard benchmarks for set packing problem [26]. As far as we know, because the railway problem instances contained confidential data related to French railways, the data were not made public. Only the random instance data are public. Therefore, we show the experiment results of DLSWCC algorithm on random instances only.

4.2. Comparison with State-of-the-Art Algorithms. Tables 1 and 2 provide the instance characteristics and the results found by CPLEX method (CPLEX), GRASP method [26] (GRASP), ACO method [27] (ACO), EA/G method [33] (EA/G), and our DLSWCC method (DLSWCC). Table 1 shows the experimental results of small-scale problem instances with 100 and 200 variables, while Table 2 shows the experimental results of medium-scale problem instances with 500 and 1000 variables.

In Tables 1 and 2, column *Var* indicates the number of variables, column *Cnst* indicates the number of constraints,

Table 1: Comparison of each algorithm on instances with 100 and 200 variables.

Instance	Characteristics					CPLEX		GRASP			ACO			EA/G			DLSWCC			
	Var	*Cnst*	Density (%)	M_one	Weight	Opt	TET	Best	Avrg	ATET	Best	Avrg	ATET	Best	Avrg	ATET	Best	Avrg	ATET	Hit
pb100rand01	100	500	2.00	2	[1-20]	**372**	2.92	**372**	**372**	1.97	**372**	**372**	3.33	**372**	**372**	0.38	**372**	**372**	0	16
pb100rand02	100	500	2.00	2	[1-1]	**34**	0.6	**34**	**34**	1.31	**34**	**34**	2	**34**	**34**	0.22	**34**	**34**	0	16
pb100rand03	100	500	3.00	4	[1-20]	**203**	7.81	**203**	**203**	1.14	**203**	**203**	2	**203**	**203**	0.37	**203**	**203**	0	16
pb100rand04	100	500	3.00	4	[1-1]	**16**	52.86	**16**	**16**	1.29	**16**	15.56	0.67	**16**	15.69	0.18	**16**	**16**	0	16
pb100rand05	100	100	2.00	2	[1-20]	**639**	0.01	**639**	**639**	0.8	**639**	**639**	1.67	**639**	**639**	0.35	**639**	**639**	0	16
pb100rand06	100	100	2.00	2	[1-1]	**64**	0.01	**64**	**64**	0.69	**64**	**64**	1	**64**	**64**	0.14	**64**	**64**	0	16
pb100rand07	100	100	2.90	4	[1-20]	**503**	0	**503**	**503**	1	**503**	**503**	1	**503**	**503**	0.38	**503**	**503**	0	16
pb100rand08	100	100	3.10	4	[1-1]	**39**	0.02	**39**	38.75	0.57	**39**	38.68	0.67	**39**	38.81	0.21	**39**	**39**	0	16
pb100rand09	100	300	2.00	2	[1-20]	**463**	0.49	**463**	**463**	1.26	**463**	**463**	1.67	**463**	**463**	0.38	**463**	**463**	0	16
pb100rand10	100	300	2.00	2	[1-1]	**40**	1.13	**40**	**40**	1.28	**40**	39.62	1	**40**	39.88	0.24	**40**	**40**	0	16
pb100rand11	100	300	3.10	4	[1-20]	**306**	0.48	**306**	**306**	0.68	**306**	**306**	1.67	**306**	**306**	0.4	**306**	**306**	0	16
pb100rand12	100	300	3.00	4	[1-1]	**23**	6.8	**23**	**23**	1.13	**23**	22.93	0.33	**23**	**23**	0.21	**23**	**23**	0	16
pb200rand01	200	1000	1.50	4	[1-20]	**416**	8760.73	**416**	415.18	7.32	**416**	415.25	27.33	**416**	**416**	1.66	**416**	**416**	0.04	16
pb200rand02	200	1000	1.50	4	[1-1]	**32**	156109.36	**32**	**32**	7.35	**32**	31.56	14.67	**32**	**32**	0.76	**32**	**32**	0.01	16
pb200rand03	200	1000	1.00	2	[1-20]	**731**	5403.23	726	722.81	10.81	729	725.12	44.33	**731**	727	1.85	**731**	**731**	0.05	16
pb200rand04	200	1000	1.00	2	[1-1]	**64**	63970.91	63	63	9.12	**64**	62.93	24.33	63	62.75	0.92	**64**	**64**	0.02	16
pb200rand05	200	1000	2.50	8	[1-20]	**184**	1211.37	**184**	**184**	4.62	**184**	182.56	16	**184**	**184**	1.07	**184**	**184**	0.05	16
pb200rand06	200	1000	2.50	8	[1-1]	**14**	8068.2	**14**	13.37	3.48	**14**	12.87	4	**14**	13.5	0.55	**14**	**14**	0.06	16
pb200rand07	200	200	1.50	4	[1-20]	**1004**	0.02	1002	1001.12	4.2	**1004**	1003.5	6.33	**1004**	1003.94	1.61	**1004**	**1004**	0.02	16
pb200rand08	200	200	1.50	4	[1-1]	**83**	0.04	**83**	82.87	2.71	**83**	82.75	2.67	**83**	82.81	0.8	**83**	**83**	0	16
pb200rand09	200	200	1.00	2	[1-20]	**1324**	0.01	**1324**	**1324**	3.75	**1324**	**1324**	7.33	**1324**	**1324**	1.31	**1324**	**1324**	0	16
pb200rand10	200	200	1.00	2	[1-1]	**118**	0.02	**118**	**118**	3.64	**118**	**118**	4	**118**	**118**	0.76	**118**	**118**	0	16
pb200rand11	200	200	2.50	8	[1-20]	**545**	0.33	**545**	544.75	2.36	**545**	**545**	4.33	**545**	**545**	1.65	**545**	**545**	0.01	16
pb200rand12	200	200	2.60	8	[1-1]	**43**	1.7	**43**	**43**	1.01	**43**	**43**	1.33	**43**	**43**	0.8	**43**	**43**	0	16
pb200rand13	200	600	1.50	4	[1-20]	**571**	830.39	**571**	566.43	6.01	**571**	568.5	20.33	**571**	570.75	1.74	**571**	**571**	0.03	16
pb200rand14	200	600	1.50	4	[1-1]	**45**	10066.91	**45**	**45**	3.92	**45**	44.43	8.67	**45**	44.62	0.75	**45**	**45**	0.01	16
pb200rand15	200	600	1.00	2	[1-20]	**926**	12.2	**926**	**926**	4.22	**926**	**926**	27	**926**	**926**	1.72	**926**	**926**	0.01	16
pb200rand16	200	600	1.00	2	[1-1]	**79**	14372.85	**79**	78.31	6.8	**79**	78.37	15.33	**79**	78.12	0.98	**79**	**79**	0	16
pb200rand17	200	600	2.50	8	[1-20]	**255**	741.52	**255**	251.31	3.61	**255**	253.25	11	**255**	254.19	1.22	**255**	**255**	0.02	16
pb200rand18	200	600	2.60	8	[1-1]	**19**	19285.06	**19**	18.06	2.35	**19**	18.12	3	**19**	18.19	0.65	**19**	**19**	0.06	16
Avrg						**305.17**	9630.27	**304.90**	304.36	3.35	**305.10**	304.53	8.63	**305.13**	304.84	0.81	**305.17**	**305.17**	0.01	16.00

Table 2: Comparison of each algorithm on instances with 500 and 1000 variables.

Instance	Characteristics					CPLEX	GRASP			ACO			EA/GC			DLSWC			
	Var	*Cnst*	Density (%)	M_one	Weight	Opt	Best	Avrg	ATET	Best	Avrg	ATET	Best	Avrg	ATET	Best	Avrg	ATET	Hit
pb500rand01	500	2500	1.20	10	[1-20]	**323***	**323**	319.38	32.08	**323**	319.87	154.67	**323**	321.31	7.03	**323**	**323**	1.07	16
pb500rand02	500	2500	1.20	10	[1-1]	24*	24	23.69	25.62	24	23.06	26.67	**25**	23.56	5.16	**25**	**25**	1.58	16
pb500rand03	500	2500	0.70	5	[1-20]	**776***	772	767.63	70.33	**776**	772.75	244.00	**776**	771.38	10.61	**776**	**776**	0.38	16
pb500rand04	500	2500	0.70	5	[1-1]	61*	61	60.13	57.30	61	60.06	69.00	**62**	60.38	6.06	**62**	**62**	0.34	16
pb500rand05	500	2500	2.20	20	[1-20]	**122**	**122**	121.5	15.48	**122**	120.62	71.00	**122**	121.56	3.92	**122**	**122**	4.71	16
pb500rand06	500	2500	2.20	20	[1-1]	**8***	**8**	**8**	12.08	**8**	7.87	99.67	**8**	7.88	2.27	**8**	**8**	4.82	16
pb500rand07	500	500	1.20	10	[1-20]	**1141**	**1141**	**1141**	13.43	**1141**	**1141**	960.00	**1141**	1139.94	9.64	**1141**	**1141**	0.13	16
pb500rand08	500	500	1.20	10	[1-1]	**89**	**89**	88.25	15.80	**89**	88.12	21.67	**89**	88.94	4.75	**89**	**89**	0.08	16
pb500rand09	500	500	0.70	5	[1-20]	**2236**	2235	2235	23.44	**2236**	2234.43	84.33	**2236**	2232.81	12.49	**2236**	**2236**	0.16	16
pb500rand10	500	500	0.70	5	[1-1]	**179**	**179**	178.06	18.20	**179**	178.31	44.67	**179**	178.56	6.12	**179**	**179**	0.06	16
pb500rand11	500	500	2.30	20	[1-20]	**424**	423	419.31	19.25	**424**	418.25	37.33	**424**	421.56	7.92	**424**	**424**	1.00	16
pb500rand12	500	500	2.20	20	[1-1]	33*	**33**	**33**	11.91	**33**	32.62	8.00	**33**	32.88	4.00	**33**	**33**	0.53	16
pb500rand13	500	1500	1.20	10	[1-20]	474*	**474**	470	32.88	**474**	468	105.00	**474**	468.56	8.90	**474**	**474**	0.51	16
pb500rand14	500	1500	1.20	10	[1-1]	37*	37	36.94	20.77	37	36.5	25.66	**38**	36.88	4.39	**38**	**38**	0.65	16
pb500rand15	500	1500	0.70	5	[1-20]	**1196**	**1196**	1186.9	59.36	**1196**	1190.93	161.67	**1196**	1185.62	10.68	**1196**	**1196**	0.15	16
pb500rand16	500	1500	0.70	5	[1-1]	**88***	**88**	86.63	36.31	**88**	86.12	60.67	**88**	87	5.12	**88**	**88**	0.15	16
pb500rand17	500	1500	2.20	20	[1-20]	**192***	**192**	191.75	18.38	**192**	188.31	57.00	**192**	191.38	5.55	**192**	**192**	2.64	16
pb500rand18	500	1500	2.20	20	[1-1]	**13***	**13**	**13**	12.03	**13**	12.43	9.00	**13**	12.81	2.85	**13**	**13**	2.66	16
pb1000rand1	1000	5000	2.60	50	[1-20]	**67**	**67**	65.5	53.50	**67**	64	117.67	**67**	**67**	10.64	**67**	**67**	129.66	16
pb1000rand2	1000	5000	2.59	50	[1-1]	**4**	**4**	3.15	39.30	**4**	3.81	15.00	**4**	3.44	3.58	**4**	**4**	131.08	16
pb1000rand3	1000	5000	0.60	10	[1-20]	**661***	649	639.5	221.20	**661**	640.5	700.67	**661**	642.81	32.92	**661**	**661**	6.41	16
pb1000rand4	1000	5000	0.60	10	[1-1]	**48***	**48**	46.83	149.70	47	45.06	108.33	**48**	46.5	18.12	**48**	**48**	6.29	16
pb1000rand5	1000	1000	2.60	50	[1-20]	**222***	**222**	217.98	64.80	**222**	219.62	85.67	**222**	217.88	23.11	**222**	**222**	46.62	16
pb1000rand6	1000	1000	2.65	50	[1-1]	**15***	14	13.68	41.40	**15**	13.37	8.00	**15**	14.06	12.03	**15**	**15**	59.52	16
pb1000rand7	1000	1000	0.58	10	[1-20]	**2260**	2222	2214.1	119.70	2248	2239.56	296.67	2253	2242.06	44.48	**2260**	**2259**	8.66	7
pb1000rand8	1000	1000	0.60	10	[1-1]	**175***	172	170.81	82.60	173	170.5	94.00	174	172.62	21.21	**175**	**175**	2.93	16
Avrg						418.00	415.69	413.53	48.73	417.42	414.45	141.00	417.81	414.98	10.91	**418.12**	**418.06**	15.88	15.65

Table 3: Comparison of each algorithm on instances with 2000 variables.

Instance	Characteristics					CPLEX	GRASP	EA/G			DLSWCC			
	Var	*Cnst*	Density (%)	M_one	Weight	Opt	Best	Best	Avrg	ATET	Best	Avrg	ATET	Hit
pb2000rand1	2000	10000	2.54	100	[1-20]	**40**	Yes	**40**	**40.00**	31.66	**40**	**40.00**	2115.64	16
pb2000rand2	2000	10000	2.55	100	[1-1]	**2**	Yes	**2**	**2.00**	10.38	**2**	**2.00**	2126.56	16
pb2000rand3	2000	10000	0.55	20	[1-20]	**478***	No	**478**	463.75	129.86	**478**	**477.06**	238.61	11
pb2000rand4	2000	10000	0.55	20	[1-1]	**32***	Yes	**32**	30.25	81.58	**32**	**31.88**	278.09	16
pb2000rand5	2000	2000	2.55	100	[1-20]	**140***	Yes	**140**	135.50	61.03	**140**	**140.00**	1419.60	16
pb2000rand6	2000	2000	2.56	100	[1-1]	**9***	Yes	**9**	8.50	34.23	**9**	**9.00**	1395.76	16
pb2000rand7	2000	2000	0.56	20	[1-20]	1784*	No	1784	1765.94	160.14	**1811**	**1810.88**	28.13	15
pb2000rand8	2000	2000	0.56	20	[1-1]	131*	No	131	130.38	73.84	**135**	**134.13**	23.19	2
Avrg						327.00		327.00	322.04	72.84	**330.88**	**330.62**	953.20	13.50

and column Density indicates the percentage of nonnull elements in the constraint matrix. Column *M_One* indicates the number of elements in the maximum set *Oj*, where $j \in \{1\ldots, m\}$, and column Weight represents the range of object weights in each instance. Note that the instances whose ranges are [1-1] are instances of the unicost set packing problem. The CPLEX solver can solve all small-scale instances (the number of variables less than or equal to 200). As shown in Table 1, column *Opt* indicates the optimal solutions found by CPLEX and column *TET* indicates the time to obtain the optimal solution. For the medium-scale instances (the number of variables equal to 500 and 1000), CPLEX cannot solve all of them. In such cases, we report the best known value in Table 2. When CPLEX cannot obtain the optimal solution, the best solution value found is marked by an asterisk ($*$). For GRASP, ACO, and EA/G methods, column Best indicates the best solution found, column *Avrg* indicates the average solution quality, and column *ATET* indicates the average execution time in seconds over 16 runs in Tables 1 and 2. The column *hit* is the number of executions reaching its best value of algorithm DLSWCC. Results of CPLEX, GRASP, and ACO methods are obtained from the literature [27]; results of EA/G approach are obtained from [33]. The bold values indicate the best solution values obtained among the compared algorithms. And the bold values in Tables 3 and 4 indicate the same meaning.

Tables 1 and 2 distinctly show that the DLSWCC method is superior to EA/G, GRASP, and ACO methods in solution quality. Out of a total of 56 instances, DLSWCC obtained the best solution that was superior to EA/G on 3 instances and the same as EA/G on the rest. In terms of average solution quality, DLSWCC is superior to EA/G on 39 instances and the same as EA/G on the rest. DLSWCC obtained the best solution that was superior to ACO on 7 instances and the same as ACO on the rest. In terms of average solution quality, DLSWCC is superior to ACO on 42 instances and the same as ACO on the rest. DLSWCC obtained the best solution that was superior to GRASP on 13 instances and the same as ACO on the rest. In terms of average solution quality, DLSWCC is superior to ACO on 32 instances and the same as GRASP on the rest. On the whole, in term of the best solution, DLSWCC method is superior to the three comparison algorithms EA/G, ACO, and GRASP on 2 instances. Similarly, in term of the average solution quality, the DLSWCC method is superior to the three comparison algorithms EA/G, ACO, and GRASP on 30 instances. More significantly, in Table 2, DLSWCC sometimes gives better values than CPLEX when CPLEX values are marked with asterisk.

Note that the system that is used to perform GRASP and ACO methods [27] is Pentium III at 800 MHz and the system that is used to perform EA/G method is Fedora 12 at 3.0 GHz which are different from the system used to perform DLSWCC. Therefore, running time cannot be compared accurately. We just make a rough comparison on the running time. Our approach is faster than EA/G, GRASP, and ACO methods on majority of the instances.

There were also 8 random instances with 2000 variables (large-scale instances), which were not used to test the ACO algorithm in literature [27]. However, in literatures [26, 33], these instances are used to test the GRASP and EA/G algorithms. Table 3 shows the results of CPLEX, GRASP, EA/G, and DLSWCC on these instances. The results for CPLEX and GRASP are obtained from [26]. When CPLEX cannot obtain the optimal solution, the best solution value found is marked by an asterisk ($*$). For GRASP algorithm, if it can obtain the best known solution, then the column corresponding to GRASP marks “yes”; otherwise, it marks “no”. Out of the 8 instances, GRASP can obtain best known values on 5 instances. DLSWCC, on the other hand, obtains as good as or better than best known values on all instances. On 2 instances, DLSWCC even improved the best known values. For the running time, our approach is a little slower than EA/G.

4.3. Comparison of Different Version of DLSWCC.

4.3. Comparison of Different Version of DLSWCC. To study the effectiveness of the dynamic scoring strategy and the weighted configuration checking strategy, we compare DLSWCC with three other alternative algorithms named DLSWCC_STATIC, DLSNOCC, and DLSECC.

In DLSWCC_STATIC, the scoring method works with a static scoring strategy, i.e., the weight of each edge will not be updated. DLSNOCC works without the weighted configuration checking strategy, i.e., it selects the vertex with the greatest score, breaking ties in favor of the oldest one during the adding procedure. DLSECC works with a straightforward extension of the configuration checking strategy instead of the weighted configuration checking strategy. We tested the three algorithms on large-scale instances over 16

Table 4: Results of algorithms DLSNOCC, DLSECC, and DLSWCC on instances with 2000 variables.

Instance	Characteristics					DLSNOCC		DLSECC		DLSWCC_STATIC		DLSWCC	
	Var	*Cnst*	Density (%)	M_one	Weight	Best	Avrg	Best	Avrg	Best	Avrg	Best	Avrg
pb2000rand1	2000	10000	2.54	100	[1-20]	**40**	**40.00**	**40**	**40.00**	**40**	**40.00**	**40**	**40**
pb2000rand2	2000	10000	2.55	100	[1-1]	**2**	**2.00**	**2**	**2.00**	**2**	**2.00**	**2**	**2**
pb2000rand3	2000	10000	0.55	20	[1-20]	**478**	469.56	**478**	476.31	475	451.88	**478**	**477.06**
pb2000rand4	2000	10000	0.55	20	[1-1]	**32**	31.75	**32**	31.81	**32**	30.94	**32**	**31.88**
pb2000rand5	2000	2000	2.55	100	[1-20]	**140**	**140.00**	**140**	**140.00**	**140**	**140.00**	**140**	**140**
pb2000rand6	2000	2000	2.56	100	[1-1]	**9**	**9.00**	**9**	**9.00**	**9**	**9.00**	**9**	**9**
pb2000rand7	2000	2000	0.56	20	[1-20]	1800	1786.44	**1811**	1810.75	1769	1713.31	**1811**	**1810.88**
pb2000rand8	2000	2000	0.56	20	[1-1]	**135**	133.25	**135**	**134.19**	133	131.06	**135**	134.13
Avrg						329.50	326.50	330.88	330.51	325.00	314.77	**330.88**	**330.62**

runs with different random seeds per instance. The results are summarised in Table 4.

From Table 4, by comparing the experimental results of algorithm DLSNOCC and algorithm DLSECC, we can see the effectiveness of the configuration checking strategy. By comparing the experimental results of algorithm DLSNECC and algorithm DLSWCC, we can see the effectiveness of the weighted configuration checking strategy. By comparing the experimental results of algorithm DLSWCC_STATIC and algorithm DLSWCC, we can see the effectiveness of the dynamic scoring strategy.

Through the above comparison, we analyse that DLSWCC algorithm is superior to EA/G algorithm, ACO algorithm, and GRASP algorithm mainly because it adopts weighted configuration checking strategy and dynamic scoring strategy, which can effectively prevent the cycling problem and avoid the algorithm falling into local minimum.

5. Conclusions

The set packing problem is a significant combinatorial optimization problem and has many real applications. In this paper, we have first researched the method of solving the SPP by encoding the problem as the maximum weighted independent set problem and tackling it with an existing maximum weighted independent set algorithm (DLSWCC). Comparing with the current best solving algorithms (EA/G, GRASP method, and ACO) for SPP, our method has yielded best results. In terms of the optimal solution and the average solution, our method has obvious advantages over the comparison methods. In terms of the solving time, our method is significantly faster than the comparison methods on majority of the instances.

In the future work, we can extend our method to solve other combinatorial optimization problems, such as dominating set problem [42, 43], generalized vertex cover problem [44], maximum diversity problem [45], maximum edge weighted clique problem [46], and multiobjective unconstrained binary quadratic programming problem [47].

Data Availability

The set packing problem data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that they have no Conflicts of Interest.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (NSFC) under Grant nos. 61806082, 61972063, and 61976050, the National Social Foundation of China under, Grant no. 19BJY246, 20BTJ062, Certificate of China Postdoctoral Science Foundation, Grant no. 2019M651208, and Jilin Education Department 13th Five-Year Science and Technology Project no. JJKH20190726KJ.

References

[1] J. Z. Peter, G. K. Leo, and P. V. Stan, “Routing trains through a railway station based on a node packing model,” *European Journal of Operational Research*, vol. 128, no. 1, pp. 14–33, 2001.

[2] D. V. Sven and V. V. Rakesh, “Combinatorial auctions: a survey,” *Informs Journal on Computing*, vol. 15, no. 3, pp. 284–309, 2003.

[3] R. Vela´squez and M. T. Melo, “A set packing approach for scheduling elective surgical procedures,” in *Operations Research Proceedings*, pp. 425–430, Springer, Berlin, Germany, 2006.

[4] E. Yuval, M. H. Magnu´s, M. Yishay, P. S. Boaz, R. Jaikumar, and R. Dror, “Online set packing,” *SIAM Journal on Computing*, vol. 41, no. 4, pp. 728–746, 2012.

[5] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, San Francisco, CA, USA, 1979.

[6] M. W. Padberg, “On the facial structure of set packing polyhedra,” *Mathematical Programming*, vol. 5, no. 1, pp. 199–215, 1973.

[7] L. Cánovas, M. Landete, and A. Maŕın, “New facets for the set packing polytope,” *Operations Research Letters*, vol. 27, no. 4, pp. 153–161, 2000.

[8] L. Cánovas, M. Landete, and A. Marín, “Facet obtaining procedures for set packing problems,” *SIAM Journal Discrete Math*, vol. 16, pp. 127–155, 2003.

[9] F. Rossi and S. Smriglio, “A set packing model for the ground holding problem in congested networks,” *European Journal of Operational Research*, vol. 131, no. 2, pp. 400–416, 2001.

[10] R. H. Kwon, G. V. Dalakouras, and C. Wang, “On a posterior evaluation of a simple greedy method for set packing,” *Optimization Letters*, vol. 2, no. 4, pp. 587–597, 2008.

[11] A. A. Kolokolov and L. A. Zaozerskaya, “On average number of iterations of some algorithms for solving the set packing problem,” *IFAC Proceedings Volumes*, vol. 42, no. 4, pp. 1510–1513, 2009.

[12] M. Landete, A. M. Rodríguez-Chía, M. Antonio, and Rodríguez-Chía, “Alternative formulations for the set packing problem and their application to the winner determination problem,” *Annals of Operations Research*, vol. 207, no. 1, pp. 137–160, 2013.

[13] R. Li, S. Hu, Y. Wang, and M. Yin, “A local search algorithm with tabu strategy and perturbation mechanism for generalized vertex cover problem,” *Neural Computing and Applications*, vol. 28, no. 7, pp. 1775–1785, 2017.

[14] R. Z. Li, S. L. Hu, J. Gao, Y. P. Zhou, Y. Y. Wang, and M. H. Yin, “GRASP for connected dominating set problems,” *Neural Computing and Applications*, vol. 28, no. 1, pp. 1059–1067, 2017.

[15] J. F. Gonçalves, M. G. C. Resende, and M. D. Costa, “A biased random-key genetic algorithm for the minimization of open stacks problem,” *International Transactions in Operational Research*, vol. 23, no. 1-2, pp. 25–46, 2016.

[16] D. Ferone, P. Festa, and M. G. C. Resende, “Hybridizations of GRASP with path relinking for the far from most string problem,” *International Transactions in Operational Research*, vol. 23, no. 3, pp. 481–506, 2016.

[17] J. S. Brandão, T. F. Noronha, M. G. C. Resende, and C. C. Ribeiro, “A biased random-key genetic algorithm for single-round divisible load scheduling,” *International Transactions in Operational Research*, vol. 22, no. 5, pp. 823–839, 2015.

[18] Y. Wang, S. Cai, J. Chen, and M. Yin, "SCCWalk: an efficient local search algorithm and its improvements for maximum weight clique problem," *Artificial Intelligence*, vol. 280, p. 103230, 2020.

[19] Y. Wang, C. Li, and M. Yin, "A two phase removing algorithm for minimum independent dominating set problem," *Applied Soft Computing*, vol. 88, p. 105949, 2020.

[20] X. Zhang, X. T. Li, and M. H. Yin, "An enhanced genetic algorithm for the distributed assembly permutation flowshop scheduling problem," *International Journal of Bio-Inspired Computation*, vol. 15, no. 2, pp. 113–124, 2020.

[21] Y. Zhou, C. Qiu, Y. Wang, M. Fan, and M. Yin, "An improved memetic algorithm for the partial vertex cover problem," *IEEE Access*, vol. 7, pp. 17389–17402, 2019.

[22] X. Zhang, X. T. Li, and J. N. Wang, "Local search algorithm with path relinking for single batch-processing machine scheduling problem," *Neural Computing and Applications*, vol. 28, no. 1, pp. 313–326, 2017.

[23] M. Rönnqvist, "A method for the cutting stock problem with different qualities," *European Journal of Operational Research*, vol. 83, no. 1, pp. 57–68, 1995.

[24] B. Chandra and M. M. Halldórsson, "Greedy local improvement and weighted set packing approximation," *Journal of Algorithms*, vol. 39, no. 2, pp. 223–240, 2001.

[25] H. C. Lau and Y. G. Goh, "An intelligent brokering system to support multi-agent Web-based 4/sup th/-party logistics," in *Proceedings 14th IEEE International Conference on Tools with Artificial Intelligence, 2002 (ICTAI 2002)*, pp. 154–161, Washington, DC, USA, November 2002.

[26] X. Delorme, X. Gandibleux, and J. Rodriguez, "GRASP for set packing problems," *European Journal of Operational Research*, vol. 153, no. 3, pp. 564–580, 2004.

[27] X. Gandibleux, X. Delorme, and V. T'Kindt, *An Ant Colony Optimisation Algorithm for the Set Packing Problem*, pp. 49–60, Springer-Verlag, Berlin, Germany, 2004.

[28] Y. Guo, A. Lim, B. Rodrigues, and Y. Zhu, "Heuristics for a bidding problem," *Computers & Operations Research*, vol. 33, no. 8, pp. 2179–2188, 2006.

[29] B. Alidaee, G. Kochenberger, K. Lewis, M. Lewis, and H. Wang, "A new approach for modeling and solving set packing problems," *European Journal of Operational Research*, vol. 186, no. 2, pp. 504–512, 2008.

[30] X. Gandibleux, J. Jorge, X. Delorme, and J. Rodriguez, *An Ant Algorithm for Measuring and Optimizing the Capacity of a Railway Infrastructure*, N. Monmarche´, F. Guinand, and P. Siarry, Eds., ISTE Ltd and John Wiley, Hoboken, NJ, USA, 2010.

[31] A. Merel, X. Gandibleux, and S. Demassey, "A collaborative combination between column generation and ant colony optimization for solving set packing problems," in *Proceedings of the The IX Metaheuristics International Conference*, pp. 25–28, Udine, Italy, January 2011.

[32] M. Sviridenko and J. Ward, "Large neighborhood local search for the maximum set packing problem," *International Colloquium on Automata, Languages, and Programming*, pp. 792–803, Springer, Berlin, Germany, 2013.

[33] S. N. Chaurasia, S. Sundar, and A. Singh, "A hybrid evolutionary approach for set packing problem," *OPSEARCH*, vol. 52, no. 2, pp. 271–284, 2015.

[34] S. N. Chaurasia, D. Jung, H. M. Lee, and J. H. Kim, "An evolutionary algorithm based hyper-heuristic for the set packing problem," in *Harmony Search and Nature Inspired Optimization Algorithms. Advances in Intelligent Systems and Computing*, N. Yadav, A. Yadav, J. Bansal, K. Deep, and J. Kim, Eds., Springer, Berlin, Germany, 2019.

[35] S. N. Chaurasia and J. H. Kim, "An evolutionary algorithm based hyper-heuristic framework for the set packing problem," *Information Sciences*, vol. 505, no. 12, pp. 1–31, 2019.

[36] M. Radman and K. Eshghi, "A framework to exploit the structure of and solve set packing problems with a semi-block-angular structure," *Computers & Industrial Engineering*, vol. 137, no. Nov., pp. 106036.113–106036.1061, 2019.

[37] R. Li, S. Hu, H. Zhang, and M. Yin, "An efficient local search framework for the minimum weighted vertex cover problem," *Information Sciences*, vol. 372, pp. 428–445, 2016.

[38] S. Cai, K. Su, and A. Sattar, "Local search with edge weighting and configuration checking heuristics for minimum vertex cover," *Artificial Intelligence*, vol. 175, no. 9-10, pp. 1672–1696, 2011.

[39] Y. Y. Wang, D. T. Ouyang, L. M. Zhang, and M. H. Yin, "A novel local search for unicast set covering problem using hyperedge configuration checking and weight diversity," *SCIENCE CHINA Information Science*, vol. 60, no. 6, Article ID 062103, 2017.

[40] C. Luo, S. W. Cai, K. L. Su, and W. Wu, "Clause states based configuration checking in local search for satisfiability," *Cybernetics, IEEE Transactions on.*vol. 45, no. 5, pp. 1014–1027, 2015.

[41] S. Cai and K. Su, "Local search for Boolean Satisfiability with configuration checking and subscore," *Artificial Intelligence*, vol. 204, pp. 75–98, 2013.

[42] Y. Wang, S. Cai, and M. Yin, "Local search for minimum weight dominating set with two-level configuration checking and frequency based scoring function," *Journal of Artificial Intelligence Research*, vol. 58, pp. 267–295, 2017.

[43] R. Li, P. S. Hu, and M. Y. YinZhou, "A novel local search algorithm for the minimum capacitated dominating set," *Journal of the Operational Research Society*, vol. 69, no. 6, pp. 849–863, 2018.

[44] S. Hu, R. Li, P. Zhao, and M. Yin, "A hybrid metaheuristic algorithm for generalized vertex cover problem," *Memetic Computing*, vol. 10, no. 2, pp. 165–176, 2018.

[45] B. Alidaee and H. Wang, "A note on heuristic approach based on UBQP formulation of the maximum diversity problem," *Journal of the Operational Research Society*, vol. 68, no. 1, pp. 102–110, 2017.

[46] R. Li, X. Wu, H. Liu, J. Wu, and M. Yin, "An efficient local search for the maximum edge weighted clique problem," *IEEE Access*, vol. 6, pp. 10743–10753, 2018.

[47] Y. Zhou, J. Wang, Z. Wu, and K. Wu, "A multi-objective tabu search algorithm based on decomposition for multi-objective unconstrained binary quadratic programming problem," *Knowledge-Based Systems*, vol. 141, no. 2, pp. 18–30, 2018.