J Comb Optim
DOI 10.1007/s10878-015-9909-3

Multi-start iterated tabu search for the minimum weight vertex cover problem

Taoqing Zhou[1] · Zhipeng Lü[1] · Yang Wang[1] · Junwen Ding[1] · Bo Peng[1]

Abstract The minimum weight vertex cover problem (MWVCP) is one of the most popular combinatorial optimization problems with various real-world applications. Given an undirected graph where each vertex is weighted, the MWVCP is to find a subset of the vertices which cover all edges of the graph and has a minimum total weight of these vertices. In this paper, we propose a multi-start iterated tabu search algorithm (MS-ITS) to tackle MWVCP. By incorporating an effective tabu search method, MS-ITS exhibits several distinguishing features, including a novel neighborhood construction procedure and a fast evaluation strategy. Extensive experiments on the set of public benchmark instances show that the proposed heuristic is very competitive with the state-of-the-art algorithms in the literature.

Keywords Minimum weight vertex cover problem · Tabu search · Fast evaluation strategy · Perturbation · Meta-heuristic

1 Introduction

In the mathematical discipline of graph theory, the minimum weight vertex cover problem (MWVCP) is a well known optimization problem with a wide range of practical applications such as in wireless communication, circuit design, and network flows. In MWVCP, we have an undirected graph $G(V, E)$ where V is the set of vertices and E is the set of edges. Each vertex $v \in V$ has a weight $w(v) \geqslant 0$. A valid solution $V^{'} \subseteq V$

✉ Bo Peng
283224262@qq.com

Zhipeng Lü
zhipeng.lui@gmail.com

[1] Smart, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, People's Republic of China

has the property that at least one of the two vertices in every edge of E belongs to subset V'. The MWVCP problem seeks to minimize the sum of the weights of vertices in a valid vertex subset V' which can cover all the edges from E.

Due to its computational complexity as an NP-complete problem (Karp et al. 1972), the research of MWVCP is concentrated on heuristic algorithms to find approximate solutions in a reasonable time. A variety of different techniques have been applied to find near optimal solutions during the last decades. It has shown that for the restricted version of the MWVCP, the so-called real-WVC, where the total weight is no larger than k and each weight is no smaller than 1, the solution can be found in time $O\,(1.3954^k + kn)$ where n represents the number of vertices in V, or in $O\,(1.3788^k + kn)$ with modestly exponential use of memory (Niedermeier and Rossmanith 2003). When weights are restricted to positive integers (integer-MVC), the problem can be solved as fast as unweighted vertex cover in time $O\,(1.2738^k + kn)$ (Chen et al. 2006). It should be noticed that the unweighted vertex cover problem can be regarded as a special case of MWVCP when the weights are equal to 1 and this related problem has already attracted much attention recently (El Ouali et al. 2014; Zhang et al. 2013). Actually, most existing algorithms for tackling the MWVCP are greedy heuristics. The first heuristic to solve the MWVCP was proposed in Chvatal (1979), which starts with an empty solution and iteratively adds the vertex with the smallest ratio between vertex weight and current degree to cover the remaining uncovered edges.

Besides, several meta-heuristics have been applied for solving the MWVCP. Singh and Gupta (2006) presented a variant genetic algorithm combined with greedy heuristic. Balachandar and Kannan (2009) proposed a meta-heuristic based on gravity. In addition, a meta-heuristic based on the ant colony optimization (ACO) approach was introduced in Shyu et al. (2004) to find approximate solutions for solving it. Afterwards, Jovanovic and Tuba (2011) improved this algorithm by introducing a pheromone correction heuristic strategy that adopts information about the best-found solution to exclude suspicious elements from it to avoid trapping into local optimum trap. Recently, Bouamama et al. (2012) addressed the MWVCP with a population-based iterated greedy algorithm (PBIG) which maintains the solutions of the population and applies, at each iteration, the basic steps of an iterated greedy algorithm to each individual of the population.

On the other hand, Tabu Search, first proposed by Glover (1989) in 1986, is a meta-heuristic search method employing local search strategy in mathematical optimization. Local search takes a potential solution to a problem and check its neighbors in order to find an improved solution. However, local search methods have a tendency to become stuck in suboptimal regions or on plateaus where many solutions are equal to each other. Tabu search enhances the performance of this technique by using memory structures that describe the visited solutions or user-provided sets of rules. This heuristic algorithm has been successfully applied to many classical problems such as circle packing problem (Fu et al. 2013; Huang et al. 2013), job shop scheduling Problem (Peng et al. 2015), course timetabling problem (Lü and Hao 2010), traveling salesman problem (Malek et al. 1989), complex network community detection (Lü and Huang 2009), and so on.

All these successful applications for solving the challenging optimization problems motivate us to explore the employment of the tabu search method to address the

MWVCP. In this vein, we develop a multi-start iterated tabu search (MS-ITS) for MWVCP to strike a better balance between the exploration and exploitation of the search space in a flexible manner. We summarize the main contributions of this study as follows:

- We propose a novel and powerful neighborhood construction procedure embedded in tabu search framework and a corresponding fast evaluation strategy to shorten the computational time.
- Two specialized tabu lists utilized in tabu search and an effective perturbation strategy are devised to enhance the intensification and diversification respectively.
- To enhance the robustness and computational efficiency of our algorithm, multiple restarting mechanism is employed.
- Tested on the widely used benchmark instances, MS-ITS can obtain highly competitive solution quality and computational efficiency with respect to the state-of-the-art algorithms in the literature .

This paper is organized as follows: Sect. 2 presents the problem description of the MWVCP. The components of our proposed MS-ITS algorithm are described in details in Sect. 3. Extensive computational evaluations and comparisons with the current state-of-the-art approaches are presented in Sect. 4. The importance of the proposed incremental evaluation technique of the proposed MS-ITS algorithm is discussed in Sect. 5. Finally, Sect. 6 concludes the paper and gives future possible research directions.

2 Description of MWVCP

In an undirected graph $G(V, E)$ where V is the set of vertices and E is the set of edges, each vertex v has a weight $w(v) \geq 0$. We define n as the number of vertices in V and m as the number of edges in E, which can represent the scale and the complexity of a real instance. The MWVCP can be formulated as follows:

$$\textbf{minimize} \quad W = \sum_{v \in V'} w(v),$$

$$\text{subject to} \quad \forall (v_i, v_j) \in E : v_i \in V' \textbf{ or } v_j \in V' \quad (1 \leqslant i,\ j \leqslant n,\ \ V' \subseteq V).$$

As described above, V' is a subset of V and it is considered as a valid solution only if it covers all edges in graph G. Moreover, if one vertex belongs to set V', we define it as *key-vertex* in our study. Otherwise, we call it *non-key-vertex*. Obviously, any vertex can be denoted as either *key-vertex* or *non-key-vertex*. Generally, this problem seeks to minimize the weights of the *key-vertices* which can cover all the edges in E.

3 Multi-start iterated tabu search

3.1 General procedure

Based on the framework of multi-start mechanism, our MS-ITS iteratively alternates between an initial solution construction procedure (line 5) and an iterated tabu search procedure (lines 6–16) to search in the given regional search space around the initial solution. More specifically, an initial solution is generated by a greedy and random method (line 5), based on which we employ a tabu search method to obtain a local optimum solution (line 8). If no further improvement can be obtained, the search turns into the perturbation phase (line 15). At this point, the algorithm repeatedly call the perturbation operator to drive the search into a new round of local search phase. This process is repeated until the best objective value has not been improved within specific threshold α (lines 7–16). Finally, we restart the previous process for a given times in order to search in different promising areas. The general description of our proposed MS-ITS algorithm is provided in Algorithm 1 and the main components are detailed in the following. Note that V'_{lbest} indicates the local optima obtained by tabu search and V'_{gbest} records the best solution found so far and N_{start} denotes the maximum restarting times of our algorithm.

Algorithm 1 Pseudo-code of MS-ITS for the MWVCP

```
1: Input: Undirected graph $G(V, E)$, $N_{start}$, $\alpha$
2: Output: The best solution $V'_{gbest}$ found so far
3: $iter \leftarrow 0$
4: while $iter < N_{start}$ do
5:   Generate initial solution $V'$ /* See Section 3.2 */
6:   $no\_improve \leftarrow 0$
7:   while $no\_improve < \alpha$ do
8:     $V'_{lbest} \leftarrow TabuSearch(V')$ /* See Section 3.3* /
9:     if $V'_{lbest}$ is better than $V'_{gbest}$ then
10:      $V'_{gbest} \leftarrow V'_{lbest}$
11:      $no\_improve \leftarrow 0$
12:    else
13:      $no\_improve \leftarrow no\_improve + 1$
14:    end if
15:    $V' \leftarrow Perturbation(V'_{lbest})$ /* See Section 3.4 */
16:  end while
17:  $iter \leftarrow iter + 1$
18: end while
```

3.2 Initial solution construction

For this problem, we generate the initial solution based on both random and greedy strategies in each restart procedure of our MS-ITS algorithm. Specifically, we first set the *key-vertex* set V' to be empty. Then, it is necessary to traverse all the edges of E and if the visited edge is un-covered, one vertex would be chosen to be added into set V'. Hence, how to choose one from two vertices of each un-covered edge is the key point. Here, we employ two random and greedy strategies to choose the *key-vertices*. More precisely, the greedy mechanism usually chooses the vertex with less weight while the random mechanism randomly chooses the vertex among the candidate vertices. Additionally, In order to strike a proper balance between intensification and diversification, the probability on selecting the greedy mechanism and the random mechanism is equal.

It should be mentioned that we also use a pure random method to generate the initial *key-vertex* set V', but do not find obvious difference for the final results. Nevertheless, an initial solution of good quality generated by the random and greedy heuristic procedure rather than the pure random procedure can save lots of computational efforts during the search. Besides, the pure greedy heuristic would fail to enhance the search diversification because of the uniform initial solutions.

3.3 Tabu search

3.3.1 Neighborhood structure

In order to search for promising and valid solutions, the proposed MS-ITS procedure employs a novel neighborhood to convert a valid solution to another valid one. Basically, it consists of several continuous basic moves, i.e., shifting one *key-vertex* into *non-key-vertex* and meanwhile shifting its adjacent vertices which are also *non-key-vertices* into *key-vertices* in order to keep the legality of the solution. More precisely, we refer to $V_r(v) = \{u|(u, v) \in E \text{ } and \text{ } u \notin V'\}$, where E is the set of edges and V' is the set of all the *key-vertices*. The neighborhood move $MV(v)$ is defined to remove a *key-vertex* v from V' and add all the associated vertices in $V_r(v)$ into V'.

As shown in Fig. 1, we give an example of the neighborhood construction process where the *key-vertices* are marked with red. The left graph indicates that the original valid solution V' has three *key-vertices* (A, B, and E). If we remove A from V' (i.e., A becomes *non-key-vertex* from *key-vertex*), the edge (A, C) will be un-covered. Then, we should add the vertex C into V' (i.e., C becomes *key-vertex* from *non-key-vertex*) to keep the validity of the solution. After that, the new solution generated by our neighborhood move is presented in the right graph of Fig. 1. Similarly, we can yield other two solutions by neighborhood moves $MV(B)$ and $MV(E)$ presented in Figs. 2 and 3, respectively. In general, the number of the candidate neighborhood solutions depends on the size of set V' (i.e., the number of *key-vertices*) in the original solution.

In the literature, very few local search strategies have been proposed to address this problem. As can be seen, all the state-of-the-art algorithms in Balachandar and Kannan (2009), Bouamama et al. (2012), Jovanovic and Tuba (2011), Shyu et al. (2004) do not utilize a local search strategy in their studies. Recently, a hybridized tabu search based on a novel local search was proposed in Voß and Fink (2012), where the neighborhood of a solution consists of those solutions that can be reached from the current solution by performing a move of adding or removing just one vertex. Based

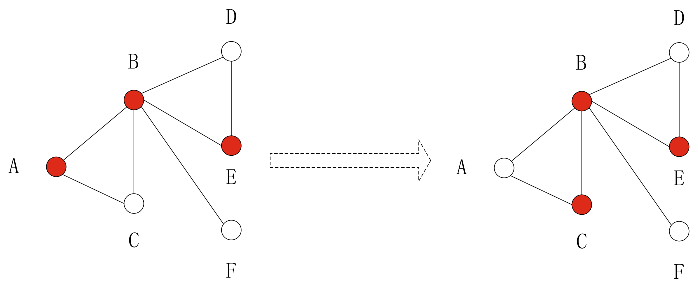

Fig. 1 Neighborhood move $MV(A)$

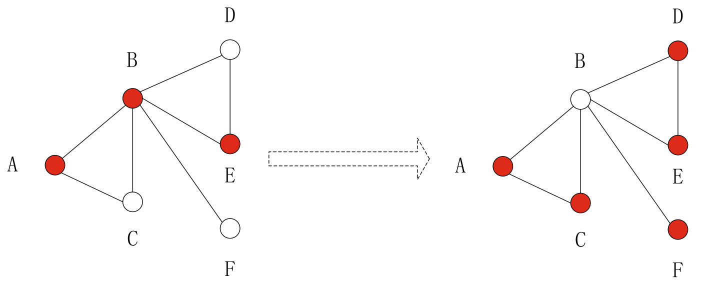

Fig. 2 Neighborhood move $MV(B)$

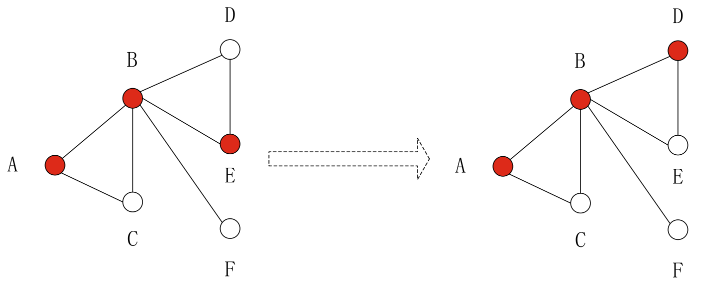

Fig. 3 Neighborhood move $MV(E)$

on this neighborhood move, the applied local search operates on a solution space that includes incomplete (infeasible) solutions, while our proposed neighborhood structure consists of moving several vertices each time. Meanwhile, we restrict the search space to be feasible during the entire search of our algorithm.

3.3.2 Fast neighborhood evaluation strategy

As mentioned above, there are a number of candidate neighborhood solutions to be evaluated at each iteration of local search. In fact, most vertices of the graph remain unchanged such that it is unnecessary to recalculate the sum of the weights of all the *key-vertices*. In order to efficiently evaluate the changes of the objective values of the candidate neighborhood moves, we propose a fast neighborhood evaluation mechanism to obtain the incremental value of each move. In our study, an evaluation function $EV(v)$ denotes the incremental objective value if $MV(v)$ is applied (i.e., removing v from V' and adding $V_r(v)$ into V'). Note that evaluation value $EV(v)$ would be set to 0 if v is a $non\text{-}key\text{-}vertex$. Specifically, the evaluation function value can be calculated as follows:

$$EV(v) = \begin{cases} \sum\limits_{u \in V_r(v)} w(u) \ - \ w(v) & \textit{if } v \in V'; \\ 0 & \textit{otherwise}. \end{cases}$$

As stated above, it is intuitive to obtain the incremental objective value with the corresponding neighborhood move through this evaluation function. Nevertheless, how to update and maintain these evaluation values is the key point in our study. In fact, the incremental evaluation values of one vertex may affect that of its adjacent vertices. Therefore, it is reasonable to update the incremental evaluation values on these affected vertices rather than all the vertices.

Algorithm 2 gives the pseudo-code of updating the incremental evaluation function (EV) values once a new solution V_n is generated by $MV(v)$ from an original solution V_o. Note that the solution in our study consists of all the chosen *key-vertices*. The specific adjustment can be operated in the following procedures: First, it is easy to observe that v is no longer a $key\text{-}vertex$, hence $EV(v)$ should be set to be 0 (line 1). Second, the evaluation values of the adjacent *key-vertices* of $v \in V_o$ should take into account the weight of vertex v (lines 2–4), while the evaluation values of the adjacent *non-key-vertices* of $v \in V_o$ need to be recalculated by the formula given above (lines 5–7). Third, the evaluation values of the *key-vertices* which are adjacent with the new added *key-vertices* in V_n should be updated by reducing the weight of the corresponding added vertex u (lines 8–12).

From Algorithm 2, one observes that the computational complexity of our fast evaluation strategy is approximately equal to $O\ (k+d^2)$, where k means the average number of *key-vertices*, and d denotes the average vertex degree of the graph. On the other hand, if we do not employ this evaluation strategy and calculate the EV values from scratch for all the *key-vertices*, the computational complexity without using our strategy is equal to $O\ (k*d)$. Moreover, the average value of d is approximately equal to $2*m/n$, while the average value of k is approximately equal to $n/2$, indicating that $k + d^2$ is smaller than $k * d$, if m is less than $n * n/4$ (the density of the graph is approximately equal to 50 %). In particular, if the graph is a sparse one, $k + d^2$ is much smaller than $k*d$. It happens that the density of most of our tested instances is less than 50 %. Therefore, our fast evaluation strategy can shorten the computational time for neighborhood evaluations to a large extent. For a clearer illustration, an experimental analysis is conducted in Sect. 5 to show the importance of this fast neighborhood evaluation strategy.

Algorithm 2 Pseudo-code of updating the incremental evaluation function values for neighborhood move $MV(v)$

```
1: $EV(v) \leftarrow 0$
2: for each $p \in P$ where $P = \{p|(p, v) \in E, p \in V_o\}$ do
3:     $EV(p) \leftarrow EV(p) + w(v)$
4: end for
5: for each $u \in U$ where $U = \{u|(u, v) \in E, u \notin V_o\}$ do
6:     Recalculate $EV(u)$
7: end for
8: for each $u \in U$ do
9:     for each $q \in Q_u$ where $Q_u = \{q|(q, u) \in E, q \in V_o - \{v\}\}$ do
10:         $EV(q) \leftarrow EV(q) - w(u)$
11:     end for
12: end for
```

3.3.3 Tabu strategy

In order to forbid the previously visited solutions to be revisited, we apply a tabu strategy to prevent local cycling. Based on our neighborhood structure, two independent tabu lists are adopted to manage the visited information as each move involves one removed *key-vertex* and several newly added *key-vertices*.

For example, if we apply move $MV(v)$, vertex v will be recorded in the first tabu list and the vertices $V_r(v)$ will be recorded in the second tabu list. Then, neighborhood move $MV(u)$ would be declared tabu if vertex u is in the first tabu list and more than γ vertices in set $V_r(u)$ are in the second tabu list. It is reasonable to utilize a subset of $V_r(u)$ to determine the tabu situation, since forbidding the whole set of $V_r(u)$ may be too strict to yield promising solutions. In general, each time we only choose the best neighborhood solution from the un-forbidden neighborhood solutions. Nevertheless, the aspiration criterion of tabu search, which allows to adopt the forbidden neighborhood move that can improve the best solution found so far, is also employed in this study. Furthermore, in order to avoid that some moves are forbidden over the entire running time, the elements in the tabu list will be cleared after a certain number of neighborhood moves (called tabu tenure λ).

3.4 Perturbation strategy

The perturbation phase aims to jump out of the search region of the incumbent solution and drive the search into other search areas. In this study, we use a unique perturbation mechanism to guide the search towards promising search spaces by removing a certain percentage (denoted as τ) of *key-vertices* and adding other necessary *key-vertices* to keep the validity of the solution. This process is similar to the initial solution construction phase other than the selection rule of adding vertices. More precisely, in the initial solution construction phase, we select the vertices based on the weight values $w(v)$ while here we prefer to select the less value of $w(v)/d'(v)$ ratio where $w(v)$ denotes the weight of candidate vertex v and $d'(v)$ denotes the number of *non-key-vertices* which are adjacent with it . Therefore, the solutions generated by this perturbation strategy would be different from the initial solution construction approach which may reduce the probability of searching the same region as before.

4 Experimental results

In this section we report extensive experimental results of applying the proposed MS-ITS algorithm to solve the standard MWVCP benchmark problem instances and compare the performance of the MS-ITS with that of the best performing algorithms in the literature.

4.1 Benchmark instances and experimental protocols

Our computational experiments are carried out on the standard benchmark instances provided by Bouamama et al. (2012). Each instance consists of an undirected and vertex-weighted graph with n vertices and m edges. According to Bouamama et al. (2012), these instances are divided into three different types:

Table 1 Parameter settings of the MS-ITS algorithm

Parameter	Description	Value
α	The consecutive iteration number of perturbation without improvement	$n/3 + 50$
β	The consecutive iteration number of tabu search without improvement	50
γ	The probability employed in tabu search strategy	$1/3$
τ	The percent of *key-vertex* set removed in the perturbation phase	$1/\text{random}(3, 6)$
λ	Tabu tenure	$20 + \text{random}(5)$
N_{start}	The maximum restart times	20

1. *Class SPI* The first set of benchmark instances (SPI) consists of the small-scale 400 instances with four different values of the number of vertices (n = 10, 15, 20, and 25).
2. *Class MPI* The second set of benchmark instances (MPI) consists of 710 middle-scale instances with six different values of the number of vertices (n = 50, 100, 150, 200, 250, and 300).
3. *Class LPI* The third set of benchmark instances (LPI) consists of 15 large-scale instances with three different values of the number of vertices (n = 500, 800, and 1000).

For the small-scale and middle-scale instances, there are 10 problem instances for each combination of n and m. Therefore, the results are presented as an average value over all the 10 instances for each combination. As mentioned in Sect. 2, there are some important parameters needed to be determined in our algorithm. Based on the comprehensive experimental results, we obtain the appropriate parameter settings as presented in Table 1.

4.2 Comparison with the best performing reference algorithms

In this section we present the performance of MS-ITS for MWVCP in comparison with the best performing algorithms, i.e., randomized gravitational emulation search algorithm (RGES) (Balachandar and Kannan 2009), ACO (Shyu et al. 2004), one improved ACO algorithm (ACO+SEE) (Jovanovic and Tuba 2011), and PBIG (Bouamama et al. 2012). Note that we execute our proposed algorithm only once on each instance of class SPI and MPI, and 10 times on each instance of class LPI.

In this study, we program our MS-ITS algorithm in C and execute it on a personal computer with AMD A6-3400M APU with 1.40 GHz, while RGES was tested in P-IV with 3.2 GHz processor running under Windows XP, and PBIG was executed on a cluster of PCs equipped with Intel X3350, 2667 MHz processors, and ACO was performed on an Intel CoreTM(2)Duo, 4.00 GHZ CPU. Obviously, the machines used by RGES and PBIG are both about twice faster than ours, and the machine used by ACO is more than three times faster than ours.

4.3 Experimental results on SPI type instances

Tables 2 and 3 provide the results obtained by MS-ITS and other state-of-the-art algorithms in terms of solution quality and computational time for the small-size problem instances SPI. As shown in these two tables, the first two columns n and m represent the number of vertices and the number of edges, respectively. The next column OPT indicates the average values of the optimum objective on each 10 instances with the same combination of n and m provided in Jovanovic and Tuba (2011). The following columns Avg and $Time$ of each algorithm present the average objective values and average computing times in seconds. Note that we marked the value which is equal to the optima in bold.

From Tables 2 and 3, we can observe that the MS-ITS and PBIG obtain very similar results in terms of both Avg and $Time$, while both algorithms could yield better solutions than others (ACO and ACO+SEE). Specifically, the ACO algorithm obtains the optimal solutions for 10 out of 20 instances in Type I and 6 out of 20 instances in Type II, and the ACO+SEE algorithm matches the optimal solutions for

Table 2 Comparison between MS-ITS and other state-of-the-art algorithms on instances of class SPI (Type I)

n	m	OPT	ACO		ACO+SEE	PBIG		MS-ITS	
			Avg	Time (s)	Avg	Avg	Time (s)	Avg	Time (s)
10	10	284.0	**284.0**	0.000	**284.0**	**284.0**	0.000	**284.0**	0.000
	20	398.7	**398.7**	0.008	**398.7**	**398.7**	0.000	**398.7**	0.000
	30	431.3	**431.3**	0.003	**431.3**	**431.3**	0.000	**431.3**	0.000
	40	508.5	**508.5**	0.003	**508.5**	**508.5**	0.000	**508.5**	0.000
15	20	441.9	**441.9**	0.005	**441.9**	**441.9**	0.000	**441.9**	0.000
	40	570.4	574.2	0.011	**570.4**	**570.4**	0.000	**570.4**	0.000
	60	726.2	729.0	0.008	**726.2**	**726.2**	0.000	**726.2**	0.000
	80	807.5	814.6	0.010	**807.5**	**807.5**	0.000	**807.5**	0.000
	100	880.0	**880.0**	0.008	**880.0**	**880.0**	0.000	**880.0**	0.000
20	20	473.0	**473.0**	0.005	**473.0**	**473.0**	0.000	**473.0**	0.000
	40	659.3	661.4	0.016	660.3	**659.3**	0.000	**659.3**	0.001
	60	861.8	**861.8**	0.014	**861.8**	**861.8**	0.000	**861.8**	0.001
	80	898.0	905.4	0.016	899.9	**898.0**	0.000	**898.0**	0.001
	100	1026.2	1026.8	0.016	**1026.2**	**1026.2**	0.000	**1026.2**	0.001
	120	1038.2	1041.5	0.017	**1038.2**	**1038.2**	0.000	**1038.2**	0.001
25	40	756.6	**756.6**	0.019	**756.6**	**756.6**	0.000	**756.6**	0.000
	80	1008.1	1009.6	0.022	**1008.1**	**1008.1**	0.000	**1008.1**	0.001
	100	1106.9	1107.4	0.025	1109.1	**1106.9**	0.000	**1106.9**	0.000
	150	1264.0	**1264.0**	0.031	**1264.0**	**1264.0**	0.000	**1264.0**	0.000
	200	1373.4	1377.7	0.030	**1373.4**	**1373.4**	0.000	**1373.4**	0.001
Avg			777.4	0.013	776.0	775.7	0.000	775.7	0.000

Table 3 Comparison between MS-ITS and other state-of-the-art algorithms on instances of class SPI (Type II)

n	m	OPT	ACO		ACO+SEE	PBIG		MS-ITS	
			Avg	Time (s)	Avg	Avg	Time (s)	Avg	Time (s)
10	10	18.8	**18.8**	0.003	**18.8**	**18.8**	0.000	**18.8**	0.000
	20	51.1	**51.1**	0.003	**51.1**	**51.1**	0.000	**51.1**	0.000
	30	127.9	**127.9**	0.003	**127.9**	**127.9**	0.000	**127.9**	0.000
	40	268.3	**268.3**	0.010	**268.3**	**268.3**	0.000	**268.3**	0.000
15	20	34.7	**34.7**	0.005	**34.7**	**34.7**	0.000	**34.7**	0.000
	40	170.5	171.5	0.010	**170.5**	**170.5**	0.000	**170.5**	0.000
	60	360.5	360.8	0.008	**360.5**	**360.5**	0.000	**360.5**	0.000
	80	697.9	698.7	0.014	**697.9**	**697.9**	0.000	**697.9**	0.000
	100	1130.4	1137.8	0.008	**1130.4**	**1130.4**	0.001	**1130.4**	0.000
20	20	32.9	33.0	0.011	**32.9**	**32.9**	0.000	**32.9**	0.001
	40	111.6	111.8	0.017	111.8	**111.6**	0.000	**111.6**	0.000
	60	254.1	254.4	0.016	**254.1**	**254.1**	0.000	**254.1**	0.001
	80	452.2	453.1	0.016	452.3	**452.2**	0.000	**452.2**	0.001
	100	775.2	**775.2**	0.016	**775.2**	**775.2**	0.000	**775.2**	0.000
	120	1123.1	1125.5	0.017	**1023.1**	**1123.1**	0.001	**1123.1**	0.001
25	40	98.7	98.8	0.025	**98.7**	**98.7**	0.000	**98.7**	0.000
	80	327.7	373.3	0.026	**373.0**	**327.7**	0.000	**327.7**	0.001
	100	595.0	595.1	0.028	595.1	**595.0**	0.000	**595.0**	0.001
	150	1289.9	1291.7	0.030	1290.9	**1289.9**	0.000	**1289.9**	0.001
	200	2709.5	2713.1	0.030	2709.5	**2709.5**	0.000	**2709.5**	0.001
Avg			534.7	0.015	533.8	533.8	0.000	533.8	0.000

17 out of 20 instances in Type I and 16 out of 20 instances in Type II, while both MS-ITS and PBIG can easily obtain the optimal solutions for all the 20 instances. Besides, both MS-ITS and PBIG can also dominate ACO algorithm on most of the instances in terms of computational time with no more than 0.001 s.

In sum, the experimental results on these small-size problem instances SPI demonstrate the high competitiveness of our proposed MT-ITS in terms of both solution quality and computational efficiency, despite the similar performance with the best performing algorithm PBIG.

4.4 Experimental results on MPI type instances

The experimental results for the problem instances of class MPI are presented in Tables 4 and 5. It is worth mentioning that the column *OPT* is not available in these two tables, as the optimal solutions are still unknown for problem instances of MPI and LPI. Additionally, the column $RGES$ in Table 5 presents the results of the RGES algorithm (Balachandar and Kannan 2009) which is just applied to perform on the Type

Table 4 Comparison between MS-ITS and other state-of-the-art algorithms on instances of class MPI (Type I)

n	m	ACO		ACO+SEE	PBIG		MS-ITS	
		Avg	Time (s)	Avg	Avg	Time (s)	Avg	Time (s)
50	50	1282.1	0.063	1280.9	**1280.0**	0.016	**1280.0**	0.001
	100	1741.1	0.083	1740.7	**1735.3**	0.007	**1735.3**	0.002
	250	2287.4	0.097	2280.6	**2272.3**	0.006	**2272.3**	0.002
	500	2679.0	0.102	2669.3	**2661.9**	0.003	**2661.9**	0.002
	750	2959.0	0.125	2957.3	**2951.0**	0.027	**2951.0**	0.002
	1000	3211.2	0.117	3199.8	**3193.7**	0.019	**3193.7**	0.003
100	100	2552.9	0.273	2544.0	2537.6	0.019	**2534.2**	0.027
	250	3626.4	0.367	3614.9	3602.7	0.057	**3601.6**	0.010
	500	4692.1	0.433	4636.4	**4600.6**	0.182	**4600.6**	0.047
	750	5076.4	0.502	5082.8	**5045.5**	0.088	**5045.5**	0.059
	1000	5534.1	0.456	5522.7	5509.4	0.084	**5508.2**	0.135
	2000	6095.7	0.589	6068.3	**6051.9**	0.501	**6051.9**	0.011
150	150	3684.9	0.691	3676.8	3667.3	0.088	**3667.0**	0.030
	250	4769.7	0.891	4754.9	4720.3	0.116	**4719.9**	0.090
	500	6224.0	1.194	6228.7	6165.7	0.294	**6165.4**	0.234
	750	7014.7	1.042	6996.3	6963.7	0.522	**6967.0**	0.118
	1000	7441.8	1.206	7383.6	7368.8	0.536	**7359.7**	0.190
	2000	8631.2	1.103	8597.2	8562.0	0.824	**8549.4**	0.317
	3000	8950.2	0.966	8940.2	**8899.8**	1.300	**8899.8**	0.080
200	250	5588.7	1.674	5572.4	5551.9	0.157	**5551.6**	0.108
	500	7259.2	2.160	7233.7	**7192.4**	0.547	7195.1	0.120
	750	8349.8	2.602	8300.3	8274.5	0.664	**8269.9**	0.283
	1000	9262.2	2.221	9208.4	9150.6	1.019	**9150.0**	0.777
	2000	10916.5	2.437	10891.1	10831.0	2.726	**10830.0**	0.650
	3000	11689.1	2.497	11680.4	11600.2	2.866	**11599.6**	0.596
250	250	6197.8	2.273	6169.2	**6148.7**	0.39	**6148.7**	0.109
	500	8538.8	4.016	8495.9	8440.7	1.344	**8438.8**	1.137
	750	9869.4	4.047	9815.5	9752.8	2.447	**9745.9**	0.521
	1000	10866.6	3.755	10791.0	10753.7	2.371	**10752.1**	0.933
	2000	12917.7	3.942	12827.0	12757.6	3.471	**12755.9**	2.298
	3000	12882.5	4.276	13830.6	13723.5	3.233	**13723.3**	1.766
	5000	14801.8	3.842	14735.9	14676.7	22.508	**14669.7**	0.648
300	300	7342.7	4.322	7326.6	7296.0	0.711	**7295.8**	0.469
	500	9517.4	5.178	9491.9	**9403.1**	1.596	9410.8	0.963
	750	11166.9	6.055	11156.5	11038.1	3.349	**11032.0**	0.698
	1000	12241.7	6.231	12163.7	12108.9	3.095	**12107.7**	0.720
	2000	14894.9	6.488	14834.6	14749.9	10.982	**14737.7**	3.230
	3000	16054.1	6.299	15910.5	15848.2	11.636	**15841.4**	0.949
	5000	17545.4	6.558	17479.8	17350.6	17.283	**17342.9**	1.605
Avg		7881.0	2.338	7848.5	7806.1	2.489	7804.0	0.511

Table 5 Comparison between MS-ITS and other state-of-the-art algorithms on instances of class MPI (Type II)

n	m	ACO		RGES	ACO+SEE	PBIG		MS-ITS	
		Avg	Time (s)	Avg	Avg	Avg	Time (s)	Avg	Time (s)
50	50	83.9	0.072	83.9	83.9	**83.7**	0.002	**83.7**	0.005
	100	276.2	0.097	276.2	274.4	**271.2**	0.003	**271.2**	0.004
	250	1886.8	0.111	1886.4	1870.3	**1853.4**	0.010	**1853.4**	0.003
	500	7915.9	0.120	7914.5	7876.7	**7825.1**	0.009	**7825.1**	0.008
	750	20134.1	0.111	20134.1	20087.6	**20079.0**	0.010	**20079.0**	0.003
100	50	67.4	0.184	67.4	**67.2**	**67.2**	0.002	**67.2**	0.010
	100	169.1	0.334	169.1	167.8	**166.6**	0.017	**166.6**	0.023
	250	901.7	0.514	890.4	895.3	**886.5**	0.065	**886.5**	0.039
	500	3726.7	0.481	3725.3	3707.0	**3693.6**	0.101	**3693.6**	0.031
	750	8754.5	0.444	8745.5	8742.3	**8680.2**	0.129	**8680.2**	0.194
150	50	**65.8**	0.292	**65.8**	65.9	**65.8**	0.001	**65.8**	0.010
	100	144.7	0.583	144.7	144.1	**144.0**	0.026	**144.0**	0.065
	250	625.7	1.387	624.4	624.8	616.0	0.187	**615.8**	0.199
	500	2375.0	1.908	2365.2	2358.6	**2331.5**	0.572	**2331.5**	0.177
	750	5799.2	1.295	5798.6	5707.0	5698.7	0.550	**5698.5**	0.244
200	50	**59.6**	0.463	**59.6**	**59.6**	**59.6**	0.001	**59.6**	0.016
	100	134.7	0.981	132.6	134.6	**134.5**	0.021	**134.5**	0.050
	250	488.7	2.413	488.4	487.9	**483.1**	0.280	484.5	0.300
	500	1843.6	3.423	1843.6	1818.7	1804.3	1.286	**1803.9**	0.375
	750	4112.8	3.600	4112.8	4077.0	4043.6	0.768	**4043.5**	0.452
250	250	423.2	3.311	423.2	421.2	**419.0**	0.476	**419.0**	0.129
	500	1457.4	5.781	1457.4	1454.3	1435.7	4.219	**1434.7**	0.869
	750	3315.9	5.983	3315.9	3289.4	3261.0	2.935	**3256.4**	0.459
	1000	6058.2	6.297	6058.2	6040.0	5989.4	7.978	**5988.2**	0.259
	2000	26149.1	4.859	26149.1	25932.1	25658.5	11.809	**25646.4**	1.698
	5000	171917.2	4.856	171917.2	171500.7	**170269.1**	37.770	**170269.1**	1.298
300	250	403.9	5.372	403.9	402.7	**399.5**	0.353	399.6	0.397
	500	1239.1	9.155	1239.1	1237.3	**1216.4**	5.439	1217.2	0.527
	750	2678.2	10.994	2678.2	2674.1	**2639.4**	6.545	2640.6	0.362
	1000	4895.5	9.045	4895.5	4867.9	4796.3	15.204	**4796.2**	0.993
	2000	21295.2	7.242	21295.2	21107.7	20891.6	10.911	**20886.4**	1.834
	3000	143243.5	6.553	143243.5	142292.6	141265.3	27.674	**141226.8**	3.418
Avg		13832.6	3.071	1383.1	13764.7	13663.4	4.230	13661.52	0.452

II instances of the class MPI. Note that the best upper bounds are marked in bold. As shown in Tables 4 and 5, our MS-ITS finds solutions no worse than the previous best known solutions except six cases of both two types instances, and has a distinguished superiority in runtime compared to other state-of-the-art algorithms. Besides, PBIG

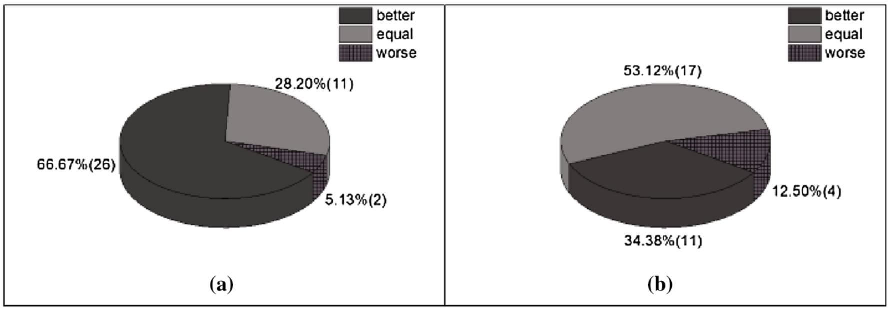

Fig. 4 Comparisons between MS-ITS and PBIG on instances MPI. **a** Type I, **b** Type II

has an average computational time of 2.489 s (referring to the last row of Table 4) and 4.230 s (the last row of Table 5), and ACO has the average computational time of 2.338 and 3.071 s, while our MS-ITS reaches the best results with an average computational time of 0.511 and 0.452 s.

As concluded in Bouamama et al. (2012), ACO, RGES, and ACO+SEE are only able to match the results of PBIG for two cases of MPI instances, which means PBIG is dominantly superior to the other three algorithms. Hence, in this paper we would compare the performance of our MS-ITS algorithm with PBIG in more details.

Figure 4 provides the numbers of the better, equal and worse solutions that our MS-ITS is able to obtain compared with the PBIG on the instances Type I and Type II, respectively. More precisely, we observe from Fig. 4a that MS-ITS can obtain better results than PBIG in 66.67 % (26 out of 39) cases, tie equal results in 28.20 % (11 out of 39) cases and worse results just only in 5.13 % (2 out of 39) cases. As shown in Fig. 4b which indicates the comparison results on instances of Type II, the proposed MS-ITS algorithm is able to obtain better results in 34.38 % (11) of all 32 cases, equal results in 53.12 % (17) while worse results in 12.50 % (4).

In sum, these experimental results demonstrate the high competitiveness of our proposed MS-ITS in terms of both solution quality and computational efficiency to solve the moderate-size problem instances (MPI) in both Type I and Type II.

4.5 Experimental results on LPI type instances

For the benchmark instances LPI, each independent combination of n and m only contains one instance that is quite different from the situation of class SPI and MPI. There are 15 different instances in total and 10 independent runs of MS-ITS are conducted on each instance of class LPI.

As shown in Table 6, column $Best$ and Avg indicate the best and average objective function values over 10 runs, respectively. Besides, column $Times(s)$ (for PBIG and MS-ITS) presents the average time to obtain the best solutions over 10 runs. Note that the best results of each instance are indicated in bold. Obviously, we can observe that the best and average results of MS-ITS are not worse than ACO+SEE for all the

Table 6 Comparison between MS-ITS and other state-of-the-art algorithms on instances of class LPI

n	m	ACO+SEE		PBIG			MS-ITS		
		Best	Avg	Best	Avg	Time (s)	Best	Avg	Time (s)
500	500	12675	12687.7	**12616**	**12620.0**	1.601	12623	12635.0	3.057
	1000	16516	16574.9	**16465**	**16470.1**	10.240	16480	16483.1	9.348
	2000	21000	21093.0	**20863**	20870.8	9.283	**20863**	**20866.9**	9.606
	5000	27294	27585.5	27318	27428.2	34.707	**27241**	**27241.0**	9.103
	10000	**29573**	29796.4	**29573**	29666.8	36.405	**29573**	**29573.0**	36.250
800	500	15049	15069.9	**15025**	**15025.0**	2.535	15046	15054.1	6.726
	1000	22792	22852.1	**22747**	22763.0	11.589	22760	**22760.0**	13.994
	2000	31680	31786.9	31355	31422.6	58.008	**31309**	**31345.7**	40.967
	5000	38830	38906.7	38665	38718.7	113.842	**38553**	**38557.1**	67.096
	10000	44499	44691.7	44396	44397.8	96.467	**44351**	**44359.9**	93.750
1000	1000	24856	24925.4	24746	**24763.1**	14.435	**24735**	24766.1	19.281
	5000	45446	45588.7	45255	45295.4	178.311	**45230**	**45256.9**	113.739
	10000	51875	52105.0	**51378**	51540.9	325.956	**51378**	**51423.0**	209.673
	15000	58394	58654.8	**58014**	58145.2	363.179	**58014**	**58068.9**	242.143
	20000	60010	60268.2	59790	59847.9	647.563	**59675**	**59719.9**	243.324
Avg		33365.9	33505.8	33213.7	33265.0	126.941	33188.7	33207.4	74.803

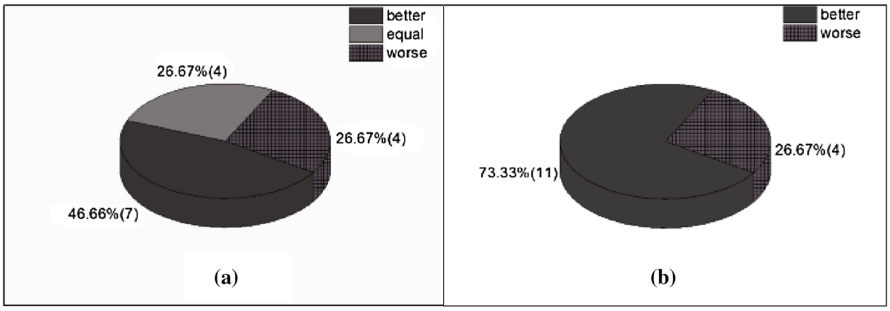

Fig. 5 Comparisons between MS-ITS and PBIG on the instances LPI. **a** Best, **b** Avg

LPI instances. In addition, Fig. 5 presents the numbers of the better, equal and worse solutions obtained by our MS-ITS algorithm compared with the PBIG algorithm. As presented in Fig. 5a, our MS-ITS algorithm is able to obtain 4 equal cases (26.67 %), 7 better cases (46.66 %) and 4 worse cases (26.67 %) of all 15 instances than PBIG with respect to the best results, while Fig. 5b implies that the MS-ITS can yield 11 better cases (73.33 %) and 4 worse cases (26.67 %) than PBIG in terms of the average results. Moreover, the average computational time of MS-ITS is about 74.803 s while PBIG costs about 126.563 s in average for the instances of class LPI. Therefore, these results indicates that our MS-ITS algorithm clearly exhibits better performance than ACO+SEE and PBIG in terms of solution quality and computational efficiency for the LPI type instances.

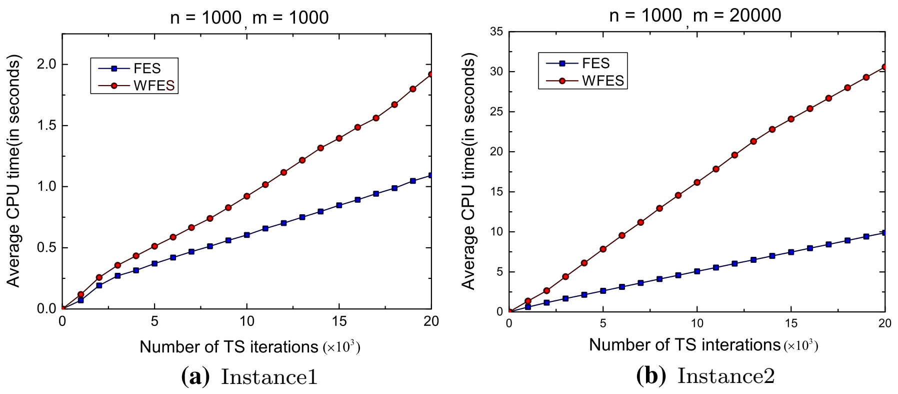

Fig. 6 Performance comparison of TS algorithm with and without the fast evaluation technique

5 Analysis

We now turn our attention to discuss and analyze the importance of the proposed incremental evaluation technique of the proposed MS-ITS algorithm.

The main ingredient of our MS-ITS algorithm is the TS procedure. For a local search method, it is particularly important to be able to rapidly determine the effect of a move on the objective function value. As described in Sect. 3.3.2, we propose a fast evaluation strategy for evaluating the neighborhood moves where we only need to maintain the EV values to help to calculate the move values of all possible candidate moves. In other words, once a move is performed, only the EV values of some specific vertices affected by the move are accordingly updated instead of recalculating the move values of all possible candidate moves.

In order to evaluate the effectiveness of this fast evaluation technique, we carry out computational experiments to compare the performance of the TS algorithm with and without using this technique on two representative instances (i.e., $n = 1000$, $m = 1000$ and $n = 1000$, $m = 10000$). The following two strategies are considered: Our fast evaluation strategy used in this paper, called fast evaluation strategy (FES) and the normal strategy without fast evaluation strategy (WFES).

For both instances, our TS algorithm is independently run for 10 times. The evolution of the average CPU time with the number of iterations is presented in Fig. 6. Specifically, for Instance 1 ($n = 1000$, $m = 1000$) and Instance 2 ($n = 1000$, $m = 10000$), it can be clearly seen that the average CPU time of TS with the FES is respectively about 2 and 3 times faster than the WFES. This experiment clearly demonstrates the importance of the FES proposed in this paper.

6 Conclusion

In this work, we propose a MS-ITS for solving the MWVCP. We apply the tabu search to find the local optimum where we first propose a problem specific neighborhood with the FES to reduce the computational time. Besides, the perturbation method has been applied to expend the search space when the search is trapped in local optimum. Furthermore, the multiple restarting mechanism is adopted to enhance the robustness and computational efficiency of the proposed MS-ITS algorithm. Experimental results demonstrate the high effectiveness of our proposed MT-ITS algorithm compared with other state-of-the-art algorithms. Additionally, by investigating some effective approximation algorithms embedded in our proposed ideas, still good outcomes may be obtained (Zhang et al. 2012).

Acknowledgments This paper is dedicated to memorizing the authors' mentor Prof. Wenqi Huang for his lifelong instruction. We thank the anonymous referee for the helpful comments to improve our paper significantly. Special thanks are given to Tao Qin and Hongyun Xu for partially implementing the code and refining the paper. The research was partially supported by the National Natural Science Foundation of China (Grant Nos. 61370183, 61100144), and the program for New Century Excellent Talents in University (NCET 2013).

References

Balachandar SR, Kannan K (2009) A meta-heuristic algorithm for vertex covering problem based on gravity. Int J Math Stat Sci 1(3):130–136

Bouamama S, Blum C, Boukerram A (2012) A population-based iterated greedy algorithm for the minimum weight vertex cover problem. Appl Soft Comput 12(6):1632–1639

Chen J, Kanj IA, Xia G (2006) Improved parameterized upper bounds for vertex cover. Math Found Comput Sci 2006:238–249

Chvatal V (1979) A greedy heuristic for the set-covering problem. Math Oper Res 4(3):233–235

El Ouali M, Fohlin H, Srivastav A (2014) An approximation algorithm for the partial vertex cover problem in hypergraphs. J Comb Optim 28:1–19

Fu Z, Huang W, Lü Z (2013) Iterated tabu search for the circular open dimension problem. Eur J Oper Res 225(2):236–243

Glover F (1989) Tabu search-part i. ORSA J Comput 1(3):190–206

Huang W, Fu Z, Xu R (2013) Tabu search algorithm combined with global perturbation for packing arbitrary sized circles into a circular container. Sci China Info Sci 56(9):1–14

Jovanovic R, Tuba M (2011) An ant colony optimization algorithm with improved pheromone correction strategy for the minimum weight vertex cover problem. Appl Soft Comput 11(8):5360–5366

Karp RM, Miller RE, Theater JW (1972) Complexity of computer computations. Plenum Press, New York

Lü Z, Hao JK (2010) Adaptive tabu search for course timetabling. Eur J Oper Res 200(1):235–244

Lü Z, Huang W (2009) Iterated tabu search for identifying community structure in complex networks. Phys Rev E 80(2):026130

Malek M, Guruswamy M, Pandya M, Owens H (1989) Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem. Ann Oper Res 21(1):59–84

Niedermeier R, Rossmanith P (2003) On efficient fixed-parameter algorithms for weighted vertex cover. J Algorithms 47(2):63–77

Peng B, Lü Z, Cheng T (2015) A tabu search/path relinking algorithm to solve the job shop scheduling problem. Computers Oper Res 53:154–164

Shyu SJ, Yin PY, Lin BM (2004) An ant colony optimization algorithm for the minimum weight vertex cover problem. Ann Oper Res 131(1–4):283–304

Singh A, Gupta AK (2006) A hybrid heuristic for the minimum weight vertex cover problem. Asia Pac J Oper Res 23(02):273–285

Voß S, Fink A (2012) A hybridized tabu search approach for the minimum weight vertex cover problem. J Heuristics 18(6):869–876

Zhang W, Wu W, Lee W, Du DZ (2012) Complexity and approximation of the connected set-cover problem. J Glob Optim 53(3):563–572

Zhang Z, Wu W, Fan L, Du DZ (2013) Minimum vertex cover in ball graphs through local search. J Glob Optim 59:1–9