IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 10, 2020, accepted January 1, 2021, date of publication January 14, 2021, date of current version March 1, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3051741

A Fast and Robust Heuristic Algorithm for the Minimum Weight Vertex Cover Problem

YANG WANG[1], ZHIPENG LÜ[1], AND ABRAHAM P. PUNNEN[2]
[1]SMART, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China
[2]Department of Mathematics, Simon Fraser University Surrey, Surrey, BC V3T 0A3, Canada

Corresponding author: Yang Wang (hust_wy@hust.edu.cn)

The work of Abraham P. Punnen was supported by the Natural Sciences and Engineering Research Council [Discovery Grant and Discovery Accelerator Supplement].

ABSTRACT The *minimum weight vertex cover problem* (MWVCP) is a fundamental combinatorial optimization problem with various real-world applications. The MWVCP seeks a vertex cover of an undirected graph such that the sum of the weights of the selected vertices is as small as possible. In this paper, we present an effective algorithm to solve the MWVCP. First, a master-apprentice evolutionary algorithm based on two individuals is conducted to enhance the diversity of solutions. Second, a hybrid tabu search combined configuration checking and solution-based tabu search is introduced to intensify local search procedure. Harnessing the power of the evolutionary strategy and a novel variant of hybrid tabu search, Master-Apprentice Evolutionary Algorithm with Hybrid Tabu Search, MAE-HTS, is presented. Results of extensive computational experiments using standard benchmark instances and other large-scale instances demonstrate the efficacy of our algorithm in terms of solution quality, running time, and robustness compared to state-of-the-art heuristics from the literature and the commercial MIP solver Gurobi. We also systematically analyze the role of each individual component of the algorithm which when worked in unison produced superior outcomes. In particular, MAE-HTS produced improved solutions for 2 out of 126 public benchmark instances with better running time. In addition, our MAE-HTS outperforms other state-of-the-art algorithms DLSWCC and NuMWVC for 72 large scale MWVCP instances by obtaining the best results for 64 ones, while other two reference algorithms can only obtain 27 best results at most.

INDEX TERMS Hybrid evolutionary algorithm, metaheuristics, minimum weight vertex cover problem, solution-based tabu search.

I. INTRODUCTION

Let $G = (V, E)$ be an undirected graph with vertex set $V = \{v_1, v_2, \ldots, v_n\}$ and edge set $E = \{e_1, e_2, \ldots e_m\}$. A weight function $w : V \rightarrow R^+ \cup \{0\}$ is also given. i.e., each vertex $v \in V$ is associated with a non-negative weight $w(v)$. Then, the *minimum weight vertex cover problem* (MWVCP) is to find a vertex cover $V' \subseteq V$ such that $\sum_{v \in V'} w(v)$ is minimized. The MWVCP is a fundamental NP-hard combinatorial optimization problem and equivalent to the maximum weight clique problem [17] on the complementary graphs. The problem also plays a vital role in numerous real-world applications [29], [33], [34]. It has been extensively analyzed by researchers from various points of view, including computational complexity [15], exact algorithms [35], [36], heuristic algorithms [2], [37], and polynomially solvable special cases [4], [25].

Although exact algorithms have gained success in small instances, they have difficulty to solve the real large instances. Thus, heuristic method becomes a popular approach for tackling such NP-hard problems. Various heuristic algorithms are available in the literature to solve the MWVCP. Up to 2012, a number of algorithms have been proposed based on greedy constructions [31], ant colony optimization approaches [14], [30] and population-based iterated greedy strategies [3]. In 2015, Zhou *et al.* proposed a multi-start iterative tabu search algorithm (MS-ITS) to solve the MWVC problem [38]. In 2016, Li *et al.* proposed a diversion local search procedure based on configuration checking (DLSWCC) to solve the problem and improve the results [19]. In 2018, Li *et al.* further improved their previous algorithm by self-adaptive removing vertex strategy [18] and proposed a

The associate editor coordinating the review of this manuscript and approving it for publication was Xujie Li.

novel local search algorithm (NuMWVC). These can be considered as the best performing algorithms for the MWVCP in the literature.

However, none of these algorithms uniformly outperformed others on standard benchmark instances. This suggests that a more careful and systematic algorithm design is necessary to achieve superior outcomes for the MWVCP. Evolutionary algorithms [1], [6], [13] and tabu search [8]–[10] are well established heuristic paradigms that have been successfully applied in solving a wide variety of hard combinatorial optimization problems of practical interest. There are unique features inherent in each of these approaches unshared by the other and hence hybridizing them to exploit their unique and dominating features could produce algorithms with superior outcomes. Hybridization of local search and genetic algorithms is not a new idea. In fact, memetic algorithms [12], [16], [23] do precisely this to reduce the probability of premature convergence of a genetic algorithm. In this paper, we investigate this line of reasoning to develop an effective and robust algorithm to solve the weighted vertex cover problem.

First, to make a better balance between diversity and time consuming, we adapt a master-apprentice evolutionary algorithm based on two individuals rather than traditional population-based evolutionary algorithms. Second, to handle the cycling problem during the local search, a hybrid tabu search which combines the configuration checking strategy and solution-based tabu search is presented. By carefully integrating various algorithmic strategies described above, a fast and robust algorithm, namely master-apprentice evolutionary algorithm with hybrid tabu search (MAE-HTS) is presented.

Systematic experimental analysis of our algorithm using four sets of 198 benchmark instances produced excellent results. More specifically, for 126 public benchmark instances, MAE-HTS obtained improved solutions for 2 instances and matched the best known solutions for the remaining 124 ones when compared with state-of-the-art reference algorithms [2], [3], [14], [19], [38] and the Gurobi MIP solver, while for 72 additional large scale instances, MAE-HTS is able to find the best solutions for 64 instances. To obtain additional insights into the distinctive features of the algorithm and computational bottlenecks, we examined the role of each of the major components of our algorithm individually through systematic experimentation. The results show that suppressing any of these major features could deteriorate the performance of the algorithm while when all the components worked in unison it resulted in a fast and robust algorithm that outperformed state-of-the-art algorithms on a variety of performance metrics.

The rest of the paper is organized as follows. A detailed description of our main algorithm is presented in Section II. Extensive computational analysis and comparisons with the current best performing algorithms are presented in Section III. Section IV investigates and analyzes the sensitivity and efficacy of some of the crucial components of our algorithm. Concluding remarks are presented in Section V.

II. MASTER-APPRENTICE EVOLUTIONARY ALGORITHM WITH HYBRID TABU SEARCH

A. MAIN FRAMEWORK OF MAE-HTS

Recall those memetic algorithms that combine evolutionary algorithms and local search are highly effective in solving various combinatorial optimization problems [11], [20], [21], [32]. The combined effect of global recombinant search and the powerful local search offers the memetic framework sufficient balance between intensification and diversification of the search process. Traditional population-based evolutionary algorithms have the disadvantage of slow convergence and high consumption of computing resources due to the large population size. The evolutionary part of our algorithm uses the master-apprentice framework and manages only two individuals (apprentices) at each iteration, where the apprentices evolve to become masters after a given number of generations (a cycle). The idea of the two-individual based evolutionary algorithm was first proposed in [22] for solving the graph coloring problem and further formalized by Ding *et al.* [5] for solving the flexible job shop scheduling problem. To absorb the essence of the history of the search process and evolution, one apprentice will be replaced by the master from the previous cycle. By adapting the two individual based formwork in our evolutionary process, we partly control and manage the issue of premature convergence, without sacrificing solution quality significantly while achieving improved running time. Any possibility of deterioration in the solution quality is taken care of by the powerful hybrid tabu search. The general architecture of our main algorithm, MAE-HTS is summarized in Algorithm 1.

Algorithm 1 Framework of MAE-HTS

Input: a graph G
Output: a best solution V_i'

1: $(G', V_s) \leftarrow Graph\_Reduction(G)$; // Section II-B
2: $S_1, S_2 \leftarrow Init()$; // Section II-C
3: $S_c, S_p, S_{best} \leftarrow Init()$;
4: $generation, cycle \leftarrow 0$;
5: **while** *stop criterion is not satisfied* **do**
6: $(S_1', S_2') \leftarrow Crossover\_Operator(S_1, S_2)$; // Section II-D
7: $S_1 \leftarrow HTS(G', S_1')$; // Section II-E
8: $S_2 \leftarrow HTS(G', S_2')$; // Section II-E
9: $S_c \leftarrow saveBest(S_1, S_2, S_c)$;
10: $S_{best} \leftarrow saveBest(S_c, S_{best})$;
11: **if** $generation\%Iter_{cycle} = 0$ **then**
12: $S_1 \leftarrow S_p$;
13: $S_p \leftarrow S_c$;
14: $S_c \leftarrow init()$;
15: $cycle \leftarrow cycle + 1$;
16: **end if**
17: $generation \leftarrow generation + 1$;
18: **end while**
19: **return** $S_{best} \cup V_s$;

The MAE-HTS algorithm is composed of the following components: A function *Graph_Reduction*() which with the input graph G produces a potentially reduced graph G' and a set V_s of vertices the status of which are fixed (i.e., in the output cover or not), a function *Init*() that generates random initial solutions, an operator *CrossOver*() to generate child solutions, and the hybrid tabu search *HTS*() to improve a given solution. We will discuss in details each of these functions and related procedures in the subsequent subsections.

For the time being, we simply want to discuss some features of the functions discussed in Algorithm 1. At the beginning, MAE-HTS invokes *Graph_Reduction*() to possibly reduce the size of the original graph and *Init*() generates two initial solutions S_1 and S_2. Then, at each generation, MAE-HTS performs the *CrossOver*() operation to generate two child solutions S'_1 and S'_2, which will be optimized by the hybrid tabu search procedure to obtain new local optimum solutions S_1 and S_2. If solutions S_1 or S_2 is better than the best solution S_{best} obtained so far, S_{best} is updated. After a given number of generations, i.e., at the end of each cycle, solution S_1 is replaced by the best solution from the previous cycle, which is stored in S_p. Meanwhile, the best solution in the current cycle is stored in S_c, which will be initialized before starting the next cycle. When the stop criterion (time limit) is met, the algorithm returns $S_{best} \cup V_s$. A high-level representation of one cycle of the MAE-HTS algorithm is given in Fig. 1.

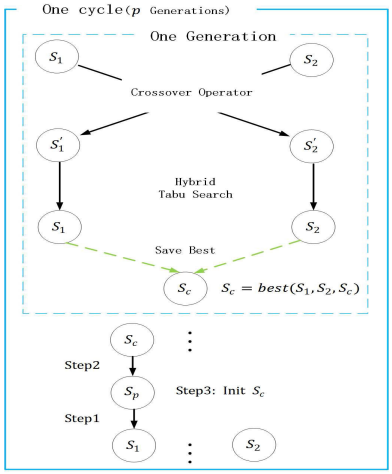


FIGURE 1. Diagram of MAE-HTS.

B. GRAPH REDUCTIONS

Graph reduction rules are sufficient conditions which if satisfied, we can determine the status of some associated vertices in an optimal solution. We first consider two simple reduction rules proposed by Wang et al in [35] for the MWVCP.

For any $v \in V$, let $N(v) = \{u|(u, v) \in E, u \in V\}$. i.e., $N(v)$ is the collection of all adjacent vertices of v in G. Recall that, for any $S \subseteq V$, $w(S) = \sum_{v \in S} w(v)$.

Adjacent rule. For any vertex $v \in V$, if $w(N(v)) \leq w(v)$, then there exists an optimal solution V' of the MWVCP such that $v \notin V'$ and $N(v) \subseteq V'$.

For any vertex v satisfying the condition of the adjacent rule, construct the graph $G' = G \setminus N(v) \setminus \{v\}$. Then, from an optimal (heuristic) solution, say U', of the MWVCP on G', an optimal (heuristic) solution V^* of the MWVCP on G can be recovered as $V^* = U' \cup N(v)$. In the adjacent rule, if for any vertex $v \in V$, if $w(N(v)) < w(v)$ (i.e., strict inequality holds), then every optimal solution V' of the MWVCP satisfies $v \notin V'$ and $N(v) \subseteq V'$.

For any vertex $v \in V$, let $N_1(v) = \{u \in N(v), |N(u)| = 1\}$. i.e., $N_1(v)$ is the collection of all adjacent vertices of v in G whose degree is one. Note that $N_1(v)$ could be an empty set.

Degree-one rule. For each vertex $v \in V$ with $N_1(v) \neq \emptyset$, if $w(v) \leq w(N_1(v))$, then there exists an optimal solution V' to the MWVCP such that $v \in V'$ and $N(v) \cap V' = \varnothing$.

For any vertex $v \in V$ satisfying the degree-one rule, construct the graph $G' = G \setminus N_1(v) \setminus \{v\}$. Then from an optimal (heuristic) solution, say U', of the MWVCP on G', an optimal solution V' of the MWVCP on G can be recovered as $V' = U' \cup \{v\}$. Further, for any vertex $v \in V$ with $N_1(v) \neq \emptyset$, if $w(v) < w(N_1(v))$ (i.e., strict inequality holds), then every optimal solution V' of the MWVCP satisfies $v \in V'$ and $N(v) \cap V' = \emptyset$.

The adjacent rule and degree one rule are special cases of the more general LP-relaxation rule. Note that the MWVCP can be formulated as an 0-1 programming problem (0-1 PP)

$$\text{Minimize} \sum_{j \in V} w_j x_j \tag{1}$$

$$\text{Subject to } x_i + x_j \geq 1 \text{ for all } (i, j) \in E \tag{2}$$

$$x_j \in \{0, 1\} \text{ for all } j \in V \tag{3}$$

where $x_j = 1$ precisely when j is in the vertex cover represented by the corresponding solution x. We denote the linear programming relaxation of 0-1 PP by LPR. It is well known that if $x^* = (x_1^*, x_2^*, \ldots, x_n^*)$ is an optimal basic feasible solution of the LPR, then $x_j^* \in \{0, 1, 1/2\}$. Let $S_0^* = \{j \in V : x_j = 0\}$ and $S_1^* = \{j \in V : x_j = 1\}$. Then it is well known [24], [26] that there exists an optimal solution $S \subseteq V$ to the MWVCP such that $S_1^* \subseteq S$ and $S \cap S_0^* = \emptyset$. This leads to yet another graph reduction procedure. Let $N^* = \cup_{v \in S_0^*} N(v)$ and $G' = G \setminus \{N^* \cup S_1^* \cup S_0^*\}$. Then from an optimal (heuristic) solution, say U', of the MWVCP on G', an optimal solution V' of the MWVCP on G can be recovered as $V' = U' \cup N^* \cup S_1^*$. Considering the time involved in solving LPR, we did not implement this graph reduction in our algorithm. However, it is not difficult to show that the two graph reductions discussed earlier are special cases of this rule.

Obviously, the graph reductions based on adjacent rule and degree one rule are dependent on the weight and degree of the adjacent vertices for each vertex in G. Once a vertex v is removed from the graph G, for each $u \in N(v)$, $w(N(u))$ and $w(N_1(u))$ with respect to the graph $G \setminus \{v\}$ can be updated

in $O(1)$ time. As mentioned in [35], the graph reduction can be applied step by step in a recursive fashion. Our graph reduction procedure is summarized in Algorithm 2.

Algorithm 2 Framework of the Graph Reductions for the MWVCP

Input: the graph $G(V, E, w)$
Output: a reduced graph G' and a vertex set V_s

set $V_s \leftarrow \varnothing$;
repeat
  set $G' \leftarrow G$;
  for each vertex $v \in V$ which satisfies the **Adjacent rule do**
    $G = G \setminus N(v) \setminus \{v\}$;
    $V_s = V_s \cup N(v)$;
  end for
  for each vertex $v \in V$ which satisfies the **Degree-one rule do**
    $G = G \setminus N_1(v) \setminus \{v\}$;
    $V_s = V_s \cup \{v\}$;
  end for
until $G' = G$;
return G' and V_s;

C. INITIAL SOLUTION

Good initial solution for each 'cycle' of the algorithm plays an important role in the effectiveness of the algorithm. For each cycle, solutions (individuals) that form the initial population are generated by a greedy randomized constructive scheme, which is adapted from the GRASP algorithm [7], [27]. First, we apply the concept of *key-vertices* introduced in [38] to denote vertices that belong to set V' while *non-key-vertices* denote the remaining ones. In fact, any vertex can be denoted as a *key-vertex* or a *non-key-vertex*. A typical construction of such a solution uses the following scheme. The algorithm performs $|V'|$ iterations where V' is the generated vertex cover. Thus the number of iterations is bounded by $|V| - 1$. We initialize $V' = \emptyset$ and designate all edges as 'uncovered'. In a general iteration, the algorithm constructs a candidate list V_c which consists of the end points of all uncovered edges by the tentative solution V'. Note that an edge is uncovered by V' if both of its end points are not in V'. For vertex v, let $\xi(v)$ denote the number of *non-key-vertices* adjacent to it. Let V_e be the subset of V_c containing z^0 vertices of V_c with smallest ratio $w(v)/\xi(v)$ (counting multiplicity) where z is a prescribed parameter and $z^0 = \min\{z, |V_c|\}$. Now choose one vertex, say v^*, from V_e with probability $\pi_v/((1 + z^0) \times z^0/2)$ for vertex v, where π_v is the ranking of the vertex v when elements of V_e are arranged in the ascending order based on the ratio. This strategy ensures that a vertex with less weight or the vertex which can cover more edges has a higher probability of being chosen. Update V' as $V' \cup \{v^*\}$ and designate all edges incident on v^* as covered. With the built in randomness, the process can generate different solutions by repeated applications. We generate two such solutions as the initial population.

Algorithm 3 Framework of the Initial Solution Generator Algorithm for the MWVCP

Initialization: set $V' = \emptyset$ and $E' = E$
while $|E'| \neq 0$ **do**
  build the candidate set V_c
  construct the elite candidates V_e
  choose a vertex v^* from V_e with a certain probability
  undate V' to $V' \cup \{v^*\}$ and E' to $E' \setminus \{e \in E' : e$ is incident on $v^*\}$
end while
output V'

D. CROSSOVER OPERATOR

The crossover operation generates offspring solutions from parent solutions. It is designed in such a way that offsprings inherit 'elite' components (qualities) from parents to a large extent. The proposed crossover operator follows this general principle and operates in two sequential steps:

- Step 1. Choose the common *key-vertices* from two parent solutions as the *key-vertices* in the offspring solution: For example, given two parents $S_1 = (1, 0, 1, 0, 1, 1)$ and $S_2 = (0, 1, 1, 1, 1, 0)$ in Fig. 2, the third and fifth vertices are the common *key-vertices* of the two parents. Hence, the third and fifth vertices will be considered as the *key-vertices* in the offspring solutions S_1' and S_2'. Since the offspring solutions generated from Step 1 may be infeasible in some cases, it is necessary to modify it to be feasible by Step 2.
- Step 2. Randomly inherit *key-vertices* to cover the remaining uncovered edges from the two parents. Specifically, as shown in Fig. 2, the offspring solution S_1' inherits v_1 as *key-vertex* from parent S_1 to cover the uncovered edge (v_1, v_2) and meanwhile select v_4 as *key-vertex* from parent S_2 to cover the uncovered edge (v_4, v_6). This step is repeated until all the uncovered edges are covered.

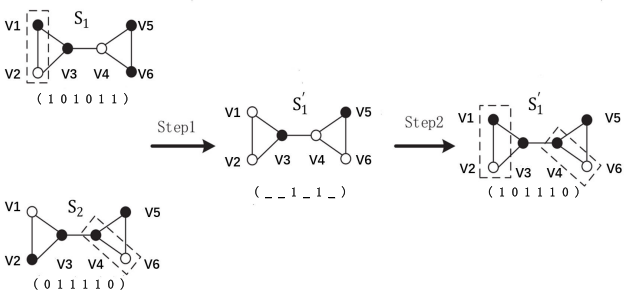


FIGURE 2. An illustration of the crossover operator.

Although the procedure of this crossover operator is relatively simple, it helps the offspring solutions to inherit elite components from parents, which is advantageous in controlling the solution quality and convergence.

E. HYBRID TABU SEARCH

Another major component of our algorithm is the enhanced local search part, which is called hybrid tabu search (HTS). The HTS procedure integrates the weighted configuration checking version of tabu search [19] with a solution-based tabu search.

Li *et al.* [19] introduced a tabu search variation with weighted configuration checking, named diversion local search based on weighted configuration checking (DLSWCC), to solve the MWVCP. The DLSWCC scheme is one of the best performing algorithms for solving the MWVCP. Although DLSWCC conducts the weighted configuration checking to reduce the issue of cycling, it could still get trapped in stagnation and falls into cycles in some situations. To overcome this undesirable situation, our HTS scheme further embeds a solution-based tabu search within the DLSWCC framework. Before getting into a formal statement of the HTS algorithm, we briefly discuss below some crucial notations, definitions, and procedures.

1) VERTEX SCORING STRATEGY

Vertex scoring is a measure used in the selection of a vertex to be added into or removed from a (partial) solution. This scoring mechanism was used by Li *et al.* [19] in their DLSWCC algorithm. For details on the application of the strategy we refer to [19]. Note that our HTS algorithm systematically adds and removes vertices from a candidate solution. After a removal operation is performed, we could get an infeasible solution which then forces back into feasibility with appropriate vertex additions. We use a dynamic scoring strategy to measure the benefit of changing the state of a vertex (i.e., to be added into a solution or removed from a solution).

Given an undirected graph $G(V, E)$ and weight function w from $V \rightarrow R^+ \cup \{0\}$, for each edge $e \in E$, we associate a weight denoted by $dynamic\_weight(e)$, which changes as the algorithm progresses. The weight $dynamic\_weight(e)$ is initially set to 1 for all $e \in E$. After adding a vertex to the current solution, if e is still uncovered by the current solution, $dynamic\_weight(e)$ will be increased by 1. For any $S \subseteq V$, let $cost(S) = \sum\{dynamic\_weight(e) : e \text{ is uncovered by } S\}$. Note that $cost(S) = 0$ if S is a vertex cover. Then, the *score of each vertex* v with respect to a (partial) solution V', denoted by $score(v, V')$, is defined as follows.

$$score(v, V') = \frac{cost(V') - cost(V'_t)}{w(v)}. \tag{4}$$

where $V'_t = V' \setminus \{v\}$ if $v \in V'$ and $V'_t = V' \cup \{v\}$ otherwise. When the underlying solution V' is obvious from the context, $score(v, V')$ is simply denoted by $score(v)$. (For example, in the algorithm $score(v)$ is calculated based on the current (partial) solution.)

2) WEIGHTED CONFIGURATION CHECKING

The concept of weighted configuration checking as a way to manage a tabu list has been described in details in [19]. We briefly discuss the concept here. Note that any tabu search algorithm manages a tabu list in some form or other to minimize the possibility of cycling. Weighted configuration checking is a way to manage a tabu list efficiently for the case of the MWVCP. We use this strategy in part of our HTS algorithm. First, we shall introduce the concept of weighted configuration.

Weighted configuration. For each vertex v, the weighted configuration of a vertex v is a two-tuple $< S, W >$, where S is a vector consisting of the states of all the vertices in $N(v)$ under the current candidate solution, and W is a vector consisting of the weights of all the incident edges of all the vertices in $N(v)$.

When selecting a vertex v to add into the current solution, if the weighted configuration of v has not been changed since its last removal from the solution, which means the circumstance of v remains stable, then v should not be added.

In details, for each vertex $v \in V$, the algorithm maintains an indicator function $wconfig : V \rightarrow \{0, 1\}$. That is, for each $v \in V$, $wconfig[v]$ denotes whether the weighted configuration of v has been changed since the last removal of v from V' and $wconfig[v] = 1$ implies that the configuration has changed. Otherwise, $wconfig[v] = 0$. At the beginning, $wconfig[v]$ is initialized to 1 for each vertex v. When removing v from V', $wconfig[v]$ will be set to 0 and for each $u \in N(v)$, $wconfig[u]$ will be set to 1. When adding v into V', for each $u \in N(v)$, $wconfig[u]$ will be set to 1. Furthermore, when $dynamic\_weight(e)$ is increasing, the weighted configuration of endpoints v and u of edge e will be set to 1 (i.e., set $wconfig[v]$ and $wconfig[u]$ to 1). Then, a vertex v is tabu when $wconfig[v] = 0$. The selected vertex v to be added into the current solution should have the property that $wconfig[v] = 1$.

3) SOLUTION-BASED TABU SEARCH

Managing cycling is a central issue in any tabu search based algorithm and researchers have used various techniques to handle this efficiently, either using general purpose methodologies or by using problem specific ones. In our HTS algorithm, we introduce a strategy which applies appropriate hash functions to record 'all visited solutions' (instead of solution attributes) to avoid cycling with high probability.

A solution V' can be expressed as a binary vector $s = (x_1, x_2, \ldots, x_n)$ where $x_i = 1$ if and only if $i \in V'$. We create a special hash function h that maps a current solution s on to the hash vector H. Each position of hash vector represents either 0 or 1 to determine whether the solution has been visited. That is, if $H(h(s)) = 1$, it indicates that solution s has been previously visited and is classified as tabu. Note that collisions could occur with a hash function. That is, two different solutions s_1 and s_2 could have the same hash value and will be mapped to the same position in H. This collision

could lead to inaccurate identification of the tabu status of candidate solutions (a non-visited solution could be forbidden incorrectly).

To effectively reduce the collision rate, we use multiple hash functions h_k $(k = 1, 2, 3)$ to keep track of the previously visited solutions. At the beginning of the tabu search, the hash vectors H_k $(k = 1, 2, 3)$ are initialized to 0, implying that no solution has been visited. Once the current feasible solution s is updated, the values at indexes $h_k(s)$ on hash vector H_k are set to 1, for $k = 1, 2, 3$. Then, a solution s is tabu if and only if all the three values $H_k[h_k(s)]$ of solution s are 1. Obviously, collisions can hardly occur simultaneously for different hash functions, resulting in reducing the probability of the misclassifying the tabu status of candidate solutions. For a given solution $s = (x_1, x_2, \ldots, x_n)$, the hash functions $h_k(s)$ is defined as:

$$h_k(s) = \left(\sum_{i=1}^{n} w_{k_i} \times x_i\right) \bmod L. \tag{5}$$

where $w_{k_i} = \lfloor i^{\gamma_k} \rfloor$, L is the length of the hash vectors, and γ_k is a parameter. The parameter values used in our algorithm are given in Section III-A.

For a solution s, we can quickly calculate the hash function value when removing or adding the ith vertex as follows:

$$h_k(s) = \begin{cases} h_k(s) - w_{k_i}, & \text{(remove } i\text{th vertex)} \\ h_k(s) + w_{k_i}, & \text{(add } i\text{th vertex)} \end{cases}$$

Obviously, the time complexity to calculate the hash value of a neighborhood solution is $O(1)$, which is computationally cheap.

Then, when adding vertex v into the current solution s, if $s \cup \{v\}$ is a feasible solution and $h_k(s \cup \{v\})(k = 1, 2, 3)$ all take 1, $s \cup \{v\}$ is regarded as a visited solution and vertex v is forbidden to be added into the current solution.

4) VERTEX SELECTION STRATEGY

Using the dynamic scoring mechanism, the weighted configuration checking and solution-based tabu search in the previous sections, we develop the vertex selection strategy. First, a concept of *age* is introduced. The *age* of a vertex is defined as the number of search steps that have elapsed since its state was last changed. Then, the vertex selection strategy is defined as follows:

- **Remove a vertex:** For vertices in the current solution s, select one vertex v with the greatest score, if there exists more than one vertex, ties are broken in favor of the oldest one, i.e., the one with the greatest value of age.
- **Add a vertex**. For vertices not in the current solution s, select the eligible vertex v with the greatest score, the eligible vertex v should take value 1 for $wconfig[v]$, and if $s \cup \{v\}$ is feasible, at least one of the three hash values should take value 0 for $h_k(s \cup \{v\})$ $(k = 1, 2, 3)$. If there exists more than one vertex, ties are broken in favor of the oldest one too.

From these two rules, we can see that HTS will delete the vertices which cover less edges and have more weight values and add the vertices which cover more edges and have less weight values on the contrary. To avoid visiting the previous encountered solutions, HTS conducts the weighted configuration checking to avoid adding recently deleted vertices and conducts the solution-based tabu search to avoid further cycling problem.

5) GENERAL PROCEDURE OF HYBRID TABU SEARCH

As described in Algorithm 4, our HTS is achieved using perturbing method to find a new solution. First, HTS records the objective value of the initial solution as IW (line 1) and initializes the hash vectors. Then, starting from a feasible solution, HTS repeatedly removes a vertex with the greatest score to an infeasible solution and records the feasible solutions in hash vectors (lines 11 and 12). Furthermore, HTS selects another vertex and the selected vertex should not be in *tabu_list*, which is used to avoid picking recently added vertices in the last iteration that are to be removed from the

Algorithm 4 Framework of Hybrid Tabu Search

Input: a reduced graph G', a initial solution V'
Output: an improved solution V'_i

1: $IW \leftarrow w(V')$;
2: initialize the improved solution $V'_i \leftarrow V'$;
3: /* Initialize the hash vector for only once in the whole MAE-HTS algorithm. */
4: initialize three hash vectors H_K to 0;
5: $Iter \leftarrow 0$;
6: **while** $Iter \leq Max\_Iter$ **do**
7: **while** V' is feasible **do**
8: **if** $w(V') < w(V'_i)$ **then**
9: $V'_i \leftarrow V'$;
10: **end if**
11: $H_k[h_k(s(V'))] \leftarrow 1$;
12: find vertex v with the greatest score in V', breaking ties in favor of the oldest one;
13: $V' \leftarrow V' \setminus \{v\}$;
14: **end while**
15: find vertex v with the greatest score in V' and not in *tabu_list*;
16: $V' \leftarrow V' \setminus \{v\}$;
17: clear *tabu_list*;
18: **while** V' is infeasible **do**
19: find eligible vertex v with the greatest score, breaking ties in favor of the oldest one;
20: **if** $w(V') + w(v) \geq IW$ **then** break;
21: $V' \leftarrow V' \cup \{v\}$;
22: add v into *tabu_list*;
23: **end while**
24: $Iter \leftarrow Iter + 1$;
25: **end while**
26: **return** V'_i;

solution (line 15). Once such a vertex is removed from the solution, HTS clears *tabu_list* (line 17). Then, HTS repeatedly tries to add an eligible vertex with the greatest score into the current solution to fix it, where the eligible vertices are determined by the weighted configuration checking and solution-based tabu search (line 19). Meanwhile, if the addition of vertex v will lead to a worse solution (i.e., a solution with objective value greater than IW), the move is forbidden and the inner loop will break (line 20). Note that HTS will record all the added vertices in one loop by *tabu_list* (line 22). When the stopping criterion (the maximum number of iterations) is reached, the improved solution V_i' is returned.

Compared to the previous DLSWCC, the main differences of our HTS are the strategy for selecting a vertex to add into the current solution (line 19) and the strategy for accepting a new solution (line 20). Our HTS conducts solution-based tabu search to avoid adding any vertex to revisit previous encountered solutions. Because HTS has the memory of visited solutions and will hardly ever cycle in a local minimum, the solutions with poor quality will have less impact during the search. Therefore, HTS can accept all solutions better than the initial solution while DLSWCC can only accept solutions better than the current best solution, i.e., HTS breaks the inner loop of adding vertices when $w(V') + w(v) \geq IW$, while DLSWCC breaks the inner loop of adding vertices when $w(V') + w(v) \geq w(V_i')$.

III. EXPERIMENTAL RESULTS

In this section, we present our results of extensive experimental analysis conducted using our MAE-HTS algorithm. We also compared our algorithm with five best performing heuristic algorithms for the MWVCP from the literature and the commercial MIP solver Gurobi.

A. BENCHMARK INSTANCES AND PARAMETER SETTING

A collection of standard benchmark instances are provided in [30]. The number of vertices in these instances varies from 10 to 1000, which are small compared to the real-world applications of the MWVCP. Therefore, 72 additional large scale instances are obtained from Network Data Repository [28], including well known DIMACS, BHOSLIB benchmarks and different types of real-life graphs, which can be categorized into biological networks, collaboration networks, interaction networks, Amazon recommend networks, scientific computation networks, social networks, Facebook networks, technological networks, and web link networks. For 72 large scale instances, the weight of each vertex is not provided. Thus, the weight of each vertex is generated as Shyu *et al.* [30] did, i.e., the vertex weights are randomly and uniformly drawn from the interval [20,120]. Note that each test instance consists of an undirected and vertex-weighted graph with n vertices and m edges. To test the performance of MAE-HTS, the following benchmark classes are used:

- **Class SPI**: the first set of benchmark (SPI) consists of 400 small instances with 4 different values of n ($n = 10$, 15, 20, and 25).
- **Class MPI**: the second set of benchmark (MPI) consists of 710 middle size instances with 6 different values of n ($n = 50$, 100, 150, 200, 250, and 300).
- **Class LPI**: the third set of benchmark (LPI) consists of 15 large scale instances with 3 different values of n ($n = 500$, 800, and 1000).
- **Class ALPI**: the fifth set of 72 large scale instances are obtained from Network Data Repository with more than 1000 vertices.[1]

For the small and middle size benchmark classes SPI and MPI, there are 10 problem instances for each combination of n and m. Therefore, the results are reported as the average value over all the 10 instances for each combination, and the proposed algorithm is executed once for these two sets of instances. For large benchmark classes LPI and ALPI, the results are obtained over 10 independent runs. All computational results of our MAE-HTS were obtained with the parameter values as presented in Table 1.[2]

TABLE 1. Parameter settings.

Parameter	Description	Value
$Iter\_cycle$	the number of generations for each cycle	5
z	a prescribed parameter for initial solution	5
Max_Iter	the number of the maximum iterations	$Min(10^6, 100*n)$
L	the length of hash vectors	$Min(10^8, n^3)$
γ_1	constant parameter in hash functions	1.3
γ_2	constant parameter in hash functions	1.5
γ_3	constant parameter in hash functions	1.8

B. COMPARISON WITH STATE-OF-THE-ART ALGORITHMS

MAE-HTS is coded in C++ and run on a PC with an AMD Ryzen 7 1700X 3.40GHz and 16GB RAM under the Windows 10 operating system. The following four best performing algorithms from the literature are selected for comparison:

- MS-ITS by Zhou *et al.* [38]
- DLSWCC by Li *et al.* [19]
- NuMWVC by Li *et al.* [18]

and an ILP solver Gurobi 9.1.

Gurobi is tested for solving the 0-1 programming problem proposed in Section II-B. Since the executable files of three reference algorithms (MS-ITS, DLSWCC and NuMWVC) have been offered by the authors. All algorithms (include Gurobi) are executed on the same PC with MAE-HTS, i.e., an AMD Ryzen 7 1700X 3.40GHz. In fact, the results obtained by the reference algorithms are better than those reported in the literature because faster computers are used in our experiments. All results provided by each algorithm are obtained by executing only once on each instance of class SPI and class

[1]available at https://github.com/HustWangYang/ALPI-class-for-MWVCP

[2]Our executable code is available at: https://github.com/HustWangYang/MAE-HTS

TABLE 2. Comparison between MAE-HTS and other state-of-the-art algorithms on instances of class SPI (Type I).

n	m	OPT	MS-ITS		DLSWCC		NuMWVC		MAE-HTS	
			Avg	*Time*	*Avg*	*Time*	*Avg*	*Time*	*Avg*	*Time*
10	10	284.0	**284.0**	0.000	**284.0**	0.000	**284.0**	0.000	**284.0**	0.000
	20	398.7	**398.7**	0.000	**398.7**	0.000	**398.7**	0.000	**398.7**	0.000
	30	431.3	**431.3**	0.000	**431.3**	0.000	**431.3**	0.000	**431.3**	0.000
	40	508.5	**508.5**	0.000	**508.5**	0.000	**431.3**	0.000	**431.3**	0.000
15	20	441.9	**441.9**	0.000	**441.9**	0.000	**441.9**	0.000	**441.9**	0.000
	40	570.4	**570.4**	0.000	**570.4**	0.000	**570.4**	0.000	**570.4**	0.000
	60	726.2	**726.2**	0.000	**726.2**	0.000	**726.2**	0.000	**726.2**	0.000
	80	807.5	**807.5**	0.000	**807.5**	0.000	**807.5**	0.000	**807.5**	0.000
	100	880.0	**880.0**	0.000	**880.0**	0.000	**880.0**	0.000	**880.0**	0.000
20	20	473.0	**473.0**	0.000	**473.0**	0.000	**473.0**	0.000	**473.0**	0.000
	40	659.3	**659.3**	0.000	**659.3**	0.000	**659.3**	0.000	**659.3**	0.000
	60	861.8	**861.8**	0.000	**861.8**	0.000	**861.8**	0.000	**861.8**	0.000
	80	898.0	**898.0**	0.000	**898.0**	0.000	**898.0**	0.000	**898.0**	0.000
	100	1026.2	**1026.2**	0.000	**1026.2**	0.000	**1026.2**	0.000	**1026.2**	0.000
	120	1038.2	**1038.2**	0.000	**1038.2**	0.000	**1038.2**	0.000	**1038.2**	0.000
25	40	756.6	**756.6**	0.000	**756.6**	0.000	**756.6**	0.000	**756.6**	0.000
	80	1008.1	**1008.1**	0.000	**1008.1**	0.000	**1008.1**	0.000	**1008.1**	0.000
	100	1106.9	**1106.9**	0.000	**1106.9**	0.000	**1106.9**	0.000	**1106.9**	0.000
	150	1264.0	**1264.0**	0.000	**1264.0**	0.000	**1264.0**	0.000	**1264.0**	0.000
	200	1373.4	**1373.4**	0.000	**1373.4**	0.000	**1373.4**	0.000	**1373.4**	0.000
AVG		**775.7**	**775.7**	0.000	**775.7**	0.000	**775.7**	0.000	**775.7**	0.000
BEST			**20**		**20**		**20**		**20**	

TABLE 3. Comparison between MAE-HTS and other state-of-the-art algorithms on instances of class SPI (Type II).

n	m	OPT	MS-ITS		DLSWCC		NuMWVC		MAE-HTS	
			Avg	*Time*	*Avg*	*Time*	*Avg*	*Time*	*Avg*	*Time*
10	10	18.8	**18.8**	0.000	**18.8**	0.000	**18.8**	0.000	**18.8**	0.000
	20	51.1	**51.1**	0.000	**51.1**	0.000	**51.1**	0.000	**51.1**	0.000
	30	127.9	**127.9**	0.000	**127.9**	0.000	**127.9**	0.000	**127.9**	0.000
	40	268.3	**268.3**	0.000	**268.3**	0.000	**268.3**	0.000	**268.3**	0.000
15	20	34.7	**34.7**	0.000	**34.7**	0.000	**34.7**	0.000	**34.7**	0.000
	40	170.5	**170.5**	0.000	**170.5**	0.000	**170.5**	0.000	**170.5**	0.000
	60	360.5	**360.5**	0.000	**360.5**	0.000	**360.5**	0.000	**360.5**	0.000
	80	697.9	**697.9**	0.000	**697.9**	0.000	**697.9**	0.000	**697.9**	0.000
	100	1130.4	**1130.4**	0.000	**1130.4**	0.000	**1130.4**	0.000	**1130.4**	0.000
20	20	32.9	**32.9**	0.000	**32.9**	0.000	**32.9**	0.000	**32.9**	0.000
	40	111.6	**111.6**	0.000	**111.6**	0.000	**111.6**	0.000	**111.6**	0.000
	60	254.1	**254.1**	0.000	**254.1**	0.000	**254.1**	0.000	**254.1**	0.000
	80	452.2	**452.2**	0.000	**452.2**	0.000	**452.2**	0.000	**452.2**	0.000
	100	775.2	**775.2**	0.000	**775.2**	0.000	**775.2**	0.000	**775.2**	0.000
	120	1123.1	**1123.1**	0.000	**1123.1**	0.000	**1123.1**	0.000	**1123.1**	0.000
25	40	98.7	**98.7**	0.000	**98.7**	0.000	**98.7**	0.000	**98.7**	0.000
	80	327.7	**327.7**	0.000	**327.7**	0.000	**327.7**	0.000	**327.7**	0.000
	100	595.0	**595.0**	0.000	**595.0**	0.000	**595.0**	0.000	**595.0**	0.000
	150	1289.9	**1289.9**	0.000	**1289.9**	0.000	**1289.9**	0.000	**1289.9**	0.000
	200	2709.5	**2709.5**	0.000	**2709.5**	0.000	**2709.5**	0.000	**2709.5**	0.000
AVG		**533.8**	**533.8**	0.000	**533.8**	0.000	**533.8**	0.000	**533.8**	0.000
BEST			**20**		**20**		**20**		**20**	

MPI, and 10 times on each instance of class LPI and class ALPI. For each instance, each algorithm is terminated upon reaching a given time limit (1000s) except for Gurobi. The time limit for Gurobi is set as 10800s. Note that if the best solution is found without reaching this time limit with normal termination code for Gurobi, it must be an optimal solution for the corresponding instance.

C. EXPERIMENTAL RESULTS ON SPI TYPE INSTANCES

The first experiment is conducted to evaluate the performance of MAE-HTS in tackling the two sets of benchmark instances (SPI Type I and Type II). For these instances, the number of vertices ranges from 10 to 25 and the number of edges ranges from 10 to 200. Tables 2 and 3 present the performance of MAE-HTS in comparison to other reference algorithms (i.e., MS-ITS, DLSWCC and NuMWVC) for the set of SPI

TABLE 4. Comparison between MAE-HTS and other state-of-the-art algorithms on instances of class MPI (Type I).

n	m	Gurobi	MS-ITS		DLSWCC		NuMWVC		MAE-HTS	
			Avg	*Time*	*Avg*	*Time*	*Avg*	*Time*	*Avg*	*Time*
50	50	**1280.0(+)**	**1280.0**	0.000	**1280.0**	0.000	**1280.0**	0.000	**1280.0**	0.000
	100	**1735.3(+)**	**1735.3**	0.000	**1735.3**	0.000	**1735.3**	0.000	**1735.3**	0.000
	250	**2272.3(+)**	**2272.3**	0.000	**2272.3**	0.000	**2272.3**	0.000	**2272.3**	0.000
	500	**2661.9(+)**	**2661.9**	0.001	**2661.9**	0.000	**2661.9**	0.000	**2661.9**	0.000
	750	**2951.0(+)**	**2951.0**	0.002	**2951.0**	0.000	**2951.0**	0.000	**2951.0**	0.000
	1000	**3193.7(+)**	**3193.7**	0.000	**3193.7**	0.000	**3193.7**	0.000	**3193.7**	0.000
100	100	**2534.2(+)**	**2534.2**	0.003	**2534.2**	0.000	**2534.2**	0.001	**2534.2**	0.000
	250	**3601.6(+)**	**3601.6**	0.009	**3601.6**	0.000	**3601.6**	0.001	**3601.6**	0.004
	500	**4600.6(+)**	**4600.6**	0.051	**4600.6**	0.000	**4600.6**	0.002	**4600.6**	0.001
	750	**5045.5(+)**	**5045.5**	0.020	**5045.5**	0.000	**5045.5**	0.000	**5045.5**	0.001
	1000	**5508.2(+)**	**5508.2**	0.004	**5508.2**	0.000	**5508.2**	0.001	**5508.2**	0.001
	2000	**6051.9(+)**	**6051.9**	0.002	**6051.9**	0.000	**6051.9**	0.001	**6051.9**	0.001
150	150	**3666.9(+)**	3667.0	0.022	**3666.9**	0.001	**3666.9**	0.001	**3666.9**	0.000
	250	**4719.9(+)**	**4719.9**	0.038	**4719.9**	0.001	**4719.9**	0.001	**4719.9**	0.001
	500	**6165.4(+)**	**6165.4**	0.100	**6165.4**	0.005	**6165.4**	0.002	**6165.4**	0.001
	750	**6956.4(+)**	**6956.4**	0.196	**6956.4**	0.003	**6956.4**	0.005	**6956.4**	0.001
	1000	**7359.7(+)**	**7359.7**	0.034	**7359.7**	0.006	**7359.7**	0.003	**7359.7**	0.002
	2000	**8549.4(+)**	**8549.4**	0.083	**8549.4**	0.010	**8549.4**	0.004	**8549.4**	0.002
	3000	**8899.8(+)**	**8899.8**	0.080	**8899.8**	0.009	**8899.8**	0.003	**8899.8**	0.002
200	250	**5551.6(+)**	**5551.6**	0.071	**5551.6**	0.005	**5551.6**	0.003	**5551.6**	0.005
	500	**7191.9(+)**	**7191.9**	0.112	**7191.9**	0.010	**7191.9**	0.011	**7191.9**	0.001
	750	**8269.9(+)**	**8269.9**	0.220	**8269.9**	0.006	**8269.9**	0.004	**8269.9**	0.002
	1000	**9145.5(+)**	**9145.5**	0.195	**9145.5**	0.012	**9145.5**	0.009	**9145.5**	0.003
	2000	**10830.0(+)**	**10830.0**	0.401	**10830.0**	0.024	**10830.0**	0.012	**10830.0**	0.004
	3000	**11595.8(+)**	**11595.8**	0.046	**11595.8**	0.015	**11595.8**	0.013	**11595.8**	0.005
250	250	**6148.7(+)**	**6148.7**	0.031	**6148.7**	0.014	**6148.7**	0.002	**6148.7**	0.004
	500	**8436.2(+)**	**8436.2**	0.292	**8436.2**	0.016	**8436.2**	0.009	**8436.2**	0.002
	750	**9745.6(+)**	9747.8	0.203	**9745.6**	0.241	**9745.6**	0.013	**9745.6**	0.004
	1000	**10751.7(+)**	10752.1	0.361	**10751.7**	0.037	**10751.7**	0.028	**10751.7**	0.015
	2000	**12751.5(+)**	12753.7	0.932	**12751.5**	0.033	**12751.5**	0.021	**12751.5**	0.013
	3000	**13723.3(+)**	**13723.3**	0.527	**13723.3**	0.057	**13723.3**	0.019	**13723.3**	0.017
	5000	**14669.7(+)**	**14669.7**	0.222	**14669.7**	0.043	**14669.7**	0.012	**14669.7**	0.007
300	300	**7295.8(+)**	7299.4	0.123	**7295.8**	0.021	**7295.8**	0.004	**7295.8**	0.002
	500	**9403.1(+)**	9404.8	0.321	**9403.1**	0.024	**9403.1**	0.013	**9403.1**	0.002
	750	**11029.3(+)**	**11029.3**	0.561	**11029.3**	0.038	**11029.3**	0.018	**11029.3**	0.005
	1000	**12098.5(+)**	12105.8	0.216	**12098.5**	0.040	**12098.5**	0.026	**12098.5**	0.003
	2000	**14732.2(+)**	14734.8	0.537	**14732.2**	0.099	**14732.2**	0.022	**14732.2**	0.008
	3000	15843.2	**15840.8**	0.383	**15840.8**	0.172	**15840.8**	0.028	**15840.8**	0.027
	5000	17354.4	**17342.9**	1.243	**17342.9**	0.195	**17342.9**	0.061	**17342.9**	0.069
AVG		7848.5	7803.3	0.196	**7802.8**	0.028	**7802.8**	0.009	**7802.8**	0.006
BEST		37	31		**39**		**39**		**39**	

instances. Columns *Avg* and *Time* show the average objective values and the average computing time in seconds by the reference algorithms MS-ITS, DLSWCC, NuMWVC and our MAE-HTS, respectively. Since optimal solutions of these instances are known, column OPT in both tables lists the optimal solution values for each instance. Row AVG presents the value averaged over one set of instances, while row BEST denotes the number of objective values obtained by the algorithm which are the best among all the algorithms. In addition, the objective value which matches the best-known results is indicated in bold.

From Tables 2 and 3, we observe that MS-ITS, DLSWCC, NuMWVC and MAE-HTS obtain the same results in terms of both *Avg* and *Time* while all these four algorithms could yield best solutions on all the instances. Small instances have no challenge for these algorithms.

D. EXPERIMENTAL RESULTS ON MPI TYPE INSTANCES

In order to further evaluate the performance of MAE-HTS, it is tested on the second set of benchmark instances (MPI type). We report the results of MAE-HTS in comparisons with other reference algorithms in tackling this set of relatively difficult instances in Tables 4 and 5. For these instances,

TABLE 5. Comparison between MAE-HTS and other state-of-the-art algorithms on instances of class MPI (Type II).

n	m	Gurobi	MS-ITS		DLSWCC		NuMWVC		MAE-HTS	
			Avg	*Time*	*Avg*	*Time*	*Avg*	*Time*	*Avg*	*Time*
50	50	**83.7(+)**	**83.7**	0.000	**83.7**	0.000	**83.7**	0.000	**83.7**	0.000
	100	**271.2(+)**	**271.2**	0.001	**271.2**	0.000	**271.2**	0.000	**271.2**	0.000
	250	**1853.4(+)**	**1853.4**	0.001	**1853.4**	0.000	**1853.4**	0.000	**1853.4**	0.000
	500	**7825.1(+)**	**7825.1**	0.002	**7825.1**	0.000	**7825.1**	0.000	**7825.1**	0.000
	750	**20079.0(+)**	**20079.0**	0.000	**20079.0**	0.000	**20079.0**	0.000	**20079.0**	0.000
100	50	**67.2(+)**	**67.2**	0.003	**67.2**	0.000	**67.2**	0.000	**67.2**	0.000
	100	**166.6(+)**	**166.6**	0.001	**166.6**	0.000	**166.6**	0.001	**166.6**	0.001
	250	**886.5(+)**	**886.5**	0.003	**886.5**	0.002	**886.5**	0.001	**886.5**	0.002
	500	**3693.6(+)**	**3693.6**	0.007	**3693.6**	0.001	**3693.6**	0.001	**3693.6**	0.001
	750	**8680.2(+)**	**8680.2**	0.031	**8680.2**	0.002	**8680.2**	0.001	**8680.2**	0.001
150	50	**65.8(+)**	**65.8**	0.002	**65.8**	0.000	**65.8**	0.000	**65.8**	0.000
	100	**144.0(+)**	**144.0**	0.008	**144.0**	0.002	**144.0**	0.001	**144.0**	0.000
	250	**615.8(+)**	616.3	0.021	**615.8**	0.002	**615.8**	0.046	**615.8**	0.001
	500	**2331.5(+)**	**2331.5**	0.049	**2331.5**	0.002	**2331.5**	0.002	**2331.5**	0.001
	750	**5698.5(+)**	5700.2	0.055	**5698.5**	0.004	**5698.5**	0.005	**5698.5**	0.005
200	50	**59.6(+)**	**59.6**	0.005	**59.6**	0.000	**59.6**	0.000	**59.6**	0.000
	100	**134.5(+)**	**134.5**	0.010	**134.5**	0.000	**134.5**	0.000	**134.5**	0.000
	250	**483.1(+)**	**483.1**	0.065	**483.1**	0.002	483.3	0.011	**483.1**	0.001
	500	**1803.9(+)**	1804	0.059	**1803.9**	0.002	1804.4	0.089	**1803.9**	0.003
	750	**4043.5(+)**	**4043.5**	0.178	**4043.5**	0.004	**4043.5**	0.074	**4043.5**	0.002
250	250	**419.0(+)**	**419.0**	0.047	**419.0**	0.012	**419.0**	0.006	**419.0**	0.000
	500	**1434.2(+)**	1434.3	0.052	**1434.2**	0.201	1435.2	0.378	**1434.2**	0.003
	750	**3256.1(+)**	**3256.1**	0.140	**3256.1**	0.115	**3256.1**	0.024	**3256.1**	0.002
	1000	**5986.1(+)**	5986.5	0.347	5986.4	0.120	5986.4	0.035	**5986.1**	0.040
	2000	**25636.5(+)**	**25636.5**	0.471	**25636.5**	0.066	**25636.5**	0.042	**25636.5**	0.022
	5000	**170269.1(+)**	**170269.1**	0.450	**170269.1**	0.006	**170269.1**	0.014	**170269.1**	0.032
300	250	**399.4(+)**	**399.4**	0.079	**399.4**	0.016	**399.4**	0.009	**399.4**	0.006
	500	**1216.4(+)**	**1216.4**	0.339	**1216.4**	0.074	1217.4	0.066	**1216.4**	0.003
	750	**2639.3(+)**	**2639.3**	0.368	**2639.3**	0.259	**2639.3**	0.197	**2639.3**	0.003
	1000	**4795.0(+)**	**4795.0**	0.275	**4795.0**	0.203	4795.5	0.156	**4795.0**	0.010
	2000	**20881.3(+)**	20885.5	0.647	**20881.3**	0.056	**20881.3**	0.037	**20881.3**	0.007
	3000	**141220.4(+)**	141223.2	0.792	**141220.4**	0.084	**141220.4**	0.132	**141220.4**	0.040
AVG		**13660.6(+)**	13660.9	0.141	**13660.6**	0.452	13660.7	0.041	**13660.6**	0.006
BEST		**32**	24		31		26		**32**	

it is hard to obtain all the optimal solutions in short time. Thus, we apply the ILP solver Gurobi to find a best solution with a reasonable time limit (10800 seconds for each instance), which are reported in column Gurobi. The optimal solution values found by Gurobi in the time limit are indicated with flag "+", and the best objective values among all the algorithms are indicated in bold.

From Tables 4 and 5, we observe that MAE-HTS outperforms other algorithms by obtaining the best results on all the instances. Indeed, MAE-HTS improves the previous best known results for one instance (with $n = 250$ and $m = 1000$ on class MPI Type II) and matches the best known solutions for the remaining 70 instances. More importantly, MAE-HTS finds the best solutions in less computational time with an average computational time of 0.006s for Type I and Type II. Compared with the ILP solver Gurobi, MAE-HTS was able to obtain all the optimal solutions and give better solutions on two MPI instances that were not solved by Gurobi to optimality.

E. EXPERIMENTAL RESULTS ON LPI TYPE INSTANCES

The third set of experiments is conducted using the LPI of instances, which are considered as the most challenging MWVCP standard instances. For this set of benchmark instances, each independent combination of n and m consists of one instance which is different from the classes SPI and MPI. Therefore, in this experiment, the proposed MAE-HTS is conducted on each instance for 10 independent runs, while the ILP solver Gurobi is still applied in the experiment with time limit 10800 seconds for each instance. The optimal objective values obtained by Gurobi are indicated with flag "+", and the best solution values among all the algorithms are indicated in bold.

Computational results of our MAE-HTS algorithm and other reference algorithms are reported in Table 6 where we report the best objective values *Best* and the average objective values *Avg* obtained over 10 independent runs for each instance. The best values of *Best* and *Avg* are indicated in bold. Table 6 discloses the superior performance of our algorithm over the reference algorithms. Indeed, MAE-HTS obtains the best results in terms of both *Best* and *Avg* for all the 15 LPI instances, which outperforms other reference algorithms. Compared with other reference algorithms, MAE-HTS is much more stable with the same values on the best and average values. MAE-HTS is significantly faster with average computational time 0.714s while MS-ITS, DLSWCC and NuMWVC have the average computational time 19.356s, 2.618 and 2.116s, respectively. In particular, our MAE-HTS could improve the best known result for one instance with $n = 800$ and $m = 2000$. Therefore, this experiment discloses the highly competitive and

TABLE 6. Comparison between MAE-HTS and other state-of-the-art algorithms on instances of class LPI.

n	m	Gurobi	MS-ITS			DLSWCC			NuMWVC			MAE-HTS		
		Val	*Best*	*Avg*	*Time*	*Best*	*Avg*	*Time*	*Best*	*Avg*	*Time*	*Best*	*Avg*	*Time*
500	500	**12616(+)**	12621	12634.5	1.904	**12616**	**12616.0**	3.304	**12616**	**12616.0**	0.200	**12616**	**12616.0**	0.004
	1000	**12465(+)**	16480	16482.6	1.408	**16465**	**16465.0**	0.142	**16465**	**16465.0**	0.554	**16465**	**16465.0**	0.026
	2000	**20863(+)**	**20863**	20863.6	3.972	**20863**	20863.6	6.739	**20863**	**20863.0**	0.978	**20863**	**20863.0**	0.031
	5000	**27241**	**27241**	**27241.0**	2.990	**27241**	**27241.0**	2.331	**27241**	**27241.0**	0.812	**27241**	**27241.0**	0.054
	10000	29848	**29573**	**29573.0**	8.747	**29573**	**29573.0**	2.614	**29573**	**29573.0**	0.304	**29573**	**29573.0**	0.034
800	500	**15025(+)**	15046	15046.0	2.807	**15025**	**15025.0**	0.199	**15025**	**15025.0**	0.184	**15025**	**15025.0**	0.077
	1000	**22747(+)**	22760	22760.0	3.405	**22747**	**22747.0**	1.738	**22747**	**22747.0**	0.320	**22747**	**22747.0**	0.008
	2000	**31283(+)**	31310	31342.2	15.919	31285	31302.0	2.482	31285	31297.5	3.431	**31283**	**31283.0**	2.184
	5000	38698	**38553**	38562.7	13.914	**38553**	38563.9	1.857	**38553**	38557.1	2.483	**38553**	**38553.0**	0.259
	10000	44473	**44351**	44363.1	31.339	**44351**	44354.0	1.523	**44351**	44355.0	2.495	**44351**	**44351.0**	1.407
1000	1000	**24723(+)**	24735	24751.5	4.321	**24723**	**24723.0**	0.551	**24723**	**24723.0**	0.401	**24723**	**24723.0**	0.003
	5000	45260	45230	45258.9	31.283	**45203**	45231.3	4.986	45215	45237.4	6.114	**45203**	**45203.0**	1.067
	10000	51690	**51378**	51454.5	50.418	**51378**	51398.8	4.958	**51378**	51389.3	3.729	**51378**	**51378.0**	0.898
	15000	58647	58014	58072.2	61.503	**57994**	57995.0	2.583	**57994**	**57994.0**	4.050	**57994**	**57994.0**	0.616
	20000	60574	59675	59735.6	56.415	**59651**	59656.4	3.262	**59651**	59655.3	5.689	**59651**	**59651.0**	4.033
AVG		33076.9	33188.7	33209.4	19.356	**33177.7**	33183.7	2.618	33178.7	33182.6	2.116	**33177.7**	**33177.7**	0.714
BEST		8	6	2		14	7		13	9		**15**	**15**	

TABLE 7. Comparison between MAE and other state-of-the-art algorithms on instances of ALPI.

Graph	n	m	DLSWCC			NuMWVC			MAE-HTS		
			Best	*Avg*	*Time*	*Best*	*Avg*	*Time*	*Best*	*Avg*	*Time*
C2000.5	2000	999836	**136452**	136644.3	1010.002	**136452**	136461.5	47.277	**136452**	**136452.0**	46.498
C2000.9	2000	1799532	139595	139625.5	3465.349	**139464**	**139464.0**	0.552	**139464**	**139464.0**	0.667
MANN-a45	1035	533115	**71923**	**71923.0**	117.112	**71923**	**71923.0**	0.214	**71923**	**71923.0**	0.000
MANN-a81	3321	5506380	234304	234304.0	776.9	**234192**	**234192.0**	0.275	**234192**	**234192.0**	0.310
keller6	3361	4619898	230608	230608.0	734.1	231054	231054.0	0.205	**230574**	**230601.2**	80.840
p-hat1500-1	1500	284923	**96959**	**96959.0**	28.060	**96959**	**96959.0**	0.968	**96959**	**96959.0**	0.097
p-hat1500-2	1500	568960	**100664**	**100664.0**	155.075	**100664**	**100664.0**	12.415	**100664**	**100664.0**	0.288
p-hat1500-3	1500	847244	**103593**	**103593.0**	753.134	**103593**	**103593.0**	1.333	**103593**	**103593.0**	0.331
frb56-25-1	1400	13422	96573	96573.0	383.261	96602	96602.0	0.152	**96479**	**96479.0**	9.227
frb56-25-2	1400	117619	**97136**	**97136.0**	308.293	97650	97705.2	0.154	**97136**	**97136.0**	0.032
frb56-25-3	1400	820644	94660	94660.0	318.132	95178	95178.0	0.154	**94638**	**94638.0**	106.242
frb56-25-4	1400	54397	95757	95757.0	316.254	95840	95840.0	0.156	**95476**	**95476.0**	1.088
frb56-25-5	1400	54397	98029	98031.1	338.183	98172	98172.0	0.152	**97978**	**97978.0**	4.836
frb59-26-1	1534	1049256	108121	108121.0	504.196	108122	108122.0	0.151	**107857**	**107857.0**	306.848
frb59-26-2	1534	1049648	106447	106447.0	540.756	106390	106390.0	0.153	**106154**	**106154.0**	30.083
frb59-26-3	1534	1049729	**102452**	102579.8	517.542	102897	102897.0	0.150	**102452**	**102452.0**	1.176
frb59-26-5	1534	1048800	104573	104639.6	514.362	104576	104618.6	0.145	**104445**	**104445.0**	7.239
frb59-26-6	1534	1049829	105725	105725.0	510.527	105583	105723.5	0.149	**105533**	**105533.0**	97.605
bio-dmela	7393	25569	168720	168747.0	64.550	168703	168705.6	29.286	**168697**	**168704.7**	470.927
bio-yeast	1458	1948	**29557**	**29557.0**	0.827	**29557**	**29557.0**	0.191	**29557**	**29557.0**	0.001
ca-AstroPh	17903	196972	760431	760798.8	216.919	**758512**	**758588.6**	276.425	759176	759272.0	403.658
ca-citeseer	227320	814134	8198352	8198575.7	567.714	**8181199**	**8182268.9**	119.648	8192170	8192470.4	742.248
ca-CondMat	21363	91286	806738	807326.8	255.180	**804022**	**804138.8**	377.597	804848	805055.3	334.742
ca-CSphd	1882	1740	**33272**	**33272.0**	1.748	**33272**	**33272.0**	0.189	**33272**	**33272.0**	0.000
ca-dblp-2010	226413	716460	7735228	7735499.9	355.806	**7715662**	**7716127.0**	101.238	7728136	7728255.9	516.009
ca-dblp-2012	317080	1049866	10464425	10464798.3	942.208	**10438989**	**10442716.5**	193.597	10454064	10454171.3	618.833
ca-Erdos992	6100	7515	**32550**	**32550.0**	0.141	**32550**	**32550.0**	0.152	**32550**	**32550.0**	0.001
ca-GrQc	4158	13422	139716	139748.9	39.388	139694	139695.7	45.142	**139693**	**139694.6**	168.317
ca-HepPh	11204	117619	432360	432550.9	136.579	430594	**430615.8**	184.248	**430518**	430899.3	382.431
ca-MathSciNet	332689	820644	8944557	8944859.5	651.722	8913041	8913490.4	136.543	**8912824**	**8912960.7**	847.960
ia-email-EU	32430	54397	**57326**	**57326.0**	2.860	**57326**	**57326.0**	0.280	**57326**	**57326.0**	0.000
ia-email-univ	1133	5451	**38443**	**38443.0**	0.851	**38443**	**38443.0**	0.822	**38443**	**38443.0**	0.028
ia-enron-large	33696	180811	813177	813480.2	373.036	**809005**	**809115.1**	500.093	809620	809822.7	681.047
ia-fb-messages	1266	6451	**37016**	**37016.0**	0.632	**37016**	**37016.0**	0.436	**37016**	**37016.0**	0.024
ia-reality	6809	7680	**5880**	**5880.0**	0.010	**5880**	**5880.0**	0.005	**5880**	**5880.0**	0.002
ia-wiki-Talk	92117	360767	1118854	1119095.7	423.125	**1108740**	1108790.5	868.007	1108764	**1108773.1**	569.748
rec-amazon	91813	125704	3050870	3051545.8	1334.450	3033106	3034416.1	997.478	**2995015**	**2995183.3**	525.404
sc-nasasrb	54870	1311227	3523027	3524019.3	1266.714	3520549	3521430.4	400.714	**3504644**	**3504777**	479.549
sc-shipsec1	140385	1707759	7987397	7988452.1	1674.865	7969093	7971518.3	467.152	**7875579**	**7876523.5**	888.297
soc-brightkite	56739	212945	1384903	1385510.5	614.468	1369082	1369431.4	684.541	**1368691**	**1368702.7**	478.993
soc-delicious	536108	1365961	5797888	5800068.8	5628.291	5750871	5759221.9	838.759	**5711358**	**5711713.5**	247.451
soc-douban	154908	327162	608827	608859.6	518.752	**608796**	**608796.0**	10.740	**608796**	**608796.0**	0.003
soc-epinions	26588	100120	631556	631934.8	272.571	626101	626128.6	324.005	**626066**	**626075.7**	710.537
soc-gowalla	196591	950327	5523350	5523903.8	2526.080	5503745	5505979.3	101.642	**5449285**	**5454981.6**	556.549
soc-slashdot	70068	358647	1449480	1449723.2	718.976	1433264	1433727.5	587.851	**1433158**	**1433201.6**	386.192
soc-twitter-follows	404719	713319	**161186**	**161186.0**	247.003	**161186**	**161186.0**	0.178	**161186**	**161186.0**	0.002
socfb-Berkeley13	22900	852419	1176301	1176862.4	530.388	1175604	1176900.9	216.025	**1173277**	**1173342.5**	255.889
socfb-CMU	6621	249959	340361	340437.2	68.700	340348	340423.7	42.409	**340209**	**340213.4**	463.651
socfb-Duke14	9885	506437	525565	525688.1	189.879	525533	525768.9	69.673	**525152**	**525169.7**	317.137
socfb-Indiana	29732	1305757	1604493	1605228.4	1119.006	1603310	1605360.3	273.519	**1599150**	**1599232.6**	572.276
socfb-MIT	6402	251230	318517	318594.0	73.355	318495	318582.2	47.264	**318385**	**318389.4**	458.666
socfb-OR	63392	816886	2461916	2463781.4	1094.764	2448921	2450014.3	786.518	**2439133**	**2439303.4**	415.601
socfb-Penn94	41536	1362220	2141044	2141785.4	1341.867	2138559	2139372.7	386.119	**2129692**	**2129770.7**	490.641
socfb-Stanford3	11586	568309	587331	587460.0	242.657	587172	587506.3	97.030	**586742**	**586757.6**	347.590
socfb-UCLA	20453	747604	1037334	1037767.7	444.288	1036682	1037344.7	187.955	**1034894**	**1034924.5**	522.520
socfb-UConn	17206	604867	905791	906142.1	302.252	906372	906646.7	140.204	**904002**	**904073.1**	413.709
socfb-UCSB37	14917	482215	767461	767990.0	226.634	767543	767986.7	120.634	**766323**	**766391.8**	843.657
socfb-UIllinois	30795	1264421	1645857	1646541.0	1079.695	1645037	1646354.0	270.752	**1639934**	**1640015.4**	569.303
socfb-Wisconsin87	23831	835946	1261878	1262382.0	518.802	1261014	1261970.3	240.069	**1257975**	**1258031.3**	474.024
tech-internet-as	40164	85123	362224	362330.1	242.812	360081	360085.6	227.229	**360076**	**360076.0**	101.423
tech-p2p-gnutella	62561	147878	1087625	1087882.7	567.099	**1085103**	1085105.4	681.142	**1085103**	**1085103.0**	136.011
tech-RL-caida	190914	607610	4926187	4926618.8	2785.616	4896788	4904101.0	989.890	**4860898**	**4861210.6**	180.821
tech-routers-rf	2113	6632	**52114**	52116.2	7.826	52115	52115.9	4.273	**52114**	**52114.0**	13.758
tech-WHOIS	7476	56943	147568	147591.3	62.579	147563	147566.3	24.174	**147559**	**147559.3**	411.475
web-arabic-2005	163598	1747269	7701067	7701573.7	1764.834	7682204	7683226.2	319.448	**7678576**	**7679167.6**	839.651
web-BerkStan	12305	19500	334037	334405.3	135.070	332835	332840.8	136.255	**332827**	**332840.0**	511.640
web-edu	3031	6474	90152	90181.9	16.695	90124	90133.4	5.866	**90108**	**90114.0**	390.409
web-google	1299	2773	**31942**	**31942.0**	2.264	**31942**	**31942.0**	0.516	**31942**	**31942.0**	0.012
web-indochina-2004	11358	47606	471007	471288.4	102.091	**469636**	**469776.4**	147.799	469872	469963.0	361.5283
web-sk-2005	121422	334419	3665046	3665222.8	78.833	3666386	3666719.8	982.408	**3654734**	**3655057.7**	336.661
web-spam	4767	37375	150351	150363.1	40.879	150342	150354.1	38.178	**150341**	**150342.6**	435.529
web-webbase-2001	16062	25593	165983	166088.0	88.659	**165701**	165707.9	58.021	**165701**	**165704.4**	251.068
AVG			1452665.2	1452927.8	592.542	1448157.1	1448717.1	189.113	**1444106.4**	**1444257.8**	285.165
BEST			17	15		27	24		**64**	**64**	

TABLE 8. Comparison between MAE-HTS with different strategies.

Graph	NuMWVC			HTS			MAE			MAE-HTS		
	Best	*Avg*	*Time*	*Best*	*Avg*	*Time*	*Best*	*Avg*	*Time*	*Best*	*Avg*	*Time*
frb56-25-1	96602	96602.0	0.152	**96479**	**96479.0**	6.731	96573	96575.9	2.152	**96479**	**96479.0**	9.227
frb56-25-2	97650	97705.2	0.154	**97136**	**97136.0**	0.040	**97136**	**97136.0**	0.033	**97136**	**97136.0**	0.032
frb56-25-3	95178	95178.0	0.154	**94638**	94644.6	10.073	94660	94660.0	2.159	**94638**	**94638.0**	106.242
frb56-25-4	95840	95840.0	0.156	**95476**	**95476.0**	5.299	**95476**	95669.3	110.723	**95476**	**95476.0**	1.088
frb56-25-5	98172	98172.0	0.152	**97978**	97982.8	10.74	98029	98033.2	25.362	**97978**	**97978.0**	4.836
frb59-26-1	108122	108122.0	0.151	**107857**	107917.2	8.701	108121	108121.0	0.036	**107857**	**107857.0**	306.848
frb59-26-2	106390	106390.0	0.153	**106154**	**106154.0**	5.404	106265	106288.4	148.931	**106154**	**106154.0**	30.083
frb59-26-3	102897	102897.0	0.150	**102452**	**102452.0**	1.213	**102452**	102494.6	106.481	**102452**	**102452.0**	1.176
frb59-26-4	104576	104618.6	0.145	**104445**	**104445.0**	2.130	104576	104623.5	36.953	**104445**	**104445.0**	7.239
frb59-26-5	105583	105723.5	0.149	**105533**	105625.6	3.977	105725	105725.0	7.540	**105533**	**105533.0**	97.605
socfb-Berkeley13	1175604	1176900.9	216.025	1173323	1173465.9	15.982	**1173269**	**1173329.2**	529.194	1173277	1173342.5	255.889
socfb-CMU	340348	340423.7	42.409	340230	340258.8	2.945	340211	340224.1	555.719	**340209**	**340213.4**	463.651
socfb-Duke14	525533	525768.9	69.673	525195	525238.6	7.807	525160	525203.7	465.840	**525152**	**525169.7**	317.137
socfb-Indiana	1603310	1605360.3	273.519	1599408	1599575.9	25.746	1599187	1599269.8	580.571	**1599150**	**1599232.6**	572.276
socfb-MIT	318495	318582.2	47.264	318401	318424.6	3.179	318387	318394.6	484.797	**318385**	**318389.4**	458.666
socfb-OR	2448921	2450014.3	786.518	2439214	2439458.3	45.719	2439248	2439376.3	208.364	**2439133**	**2439303.4**	415.601
socfb-Penn94	2138559	2139372.7	386.119	2129897	2130057.5	32.965	2129724	2129811.4	454.033	**2129692**	**2129770.7**	490.641
socfb-Stanford3	587172	587506.3	97.030	586778	586821.8	7.755	586744	586762.0	439.417	**586742**	**586757.6**	347.590
socfb-UCLA	1036682	1037344.7	187.955	1034916	1035031.9	11.718	1034906	1034940.0	474.586	**1034894**	**1034924.5**	522.520
socfb-UConn	906372	906646.7	140.204	904141	904245.6	8.499	904005	904072.2	650.338	**904002**	**904073.1**	413.709
socfb-UCSB37	767543	767986.7	120.634	766490	766584.2	11.299	766325	766414.6	531.335	**766323**	**766391.8**	843.657
socfb-UIllinois	1645037	1646354.0	270.752	1640115	1640287.4	25.097	1639943	1640071.3	683.668	**1639934**	**1640015.4**	569.303
socfb-Wisconsin87	1261014	1261970.3	240.069	1258250	1258277.0	16.860	1258050	1258092.4	658.095	**1257975**	**1258031.3**	474.024
BEST	0	0		10	6		4	2		**22**	**22**	

TABLE 9. Experiments for parameter settings.

Graph	$\alpha = 1,\beta = 1$		$\alpha = 1,\beta = 0.6$		$\alpha = 1,\beta = 2$		$\alpha = 0.6,\beta = 1$		$\alpha = 2,\beta = 1$	
	Best	*Avg*	*Best*	*Avg*	*Best*	*Avg*	*Best*	*Avg*	*Best*	*Avg*
frb59-26-1	**107857**	**107857.0**	**107857**	**107857.0**	**107857**	**107857.0**	**107857**	**107857.0**	**107857**	**107857.0**
frb59-26-2	**106154**	**106154.0**	**106154**	**106154.0**	**106154**	**106154.0**	**106154**	**106154.0**	**106154**	**106154.0**
frb59-26-3	**102452**	**102452.0**	**102452**	**102452.0**	**102452**	**102452.0**	**102452**	**102452.0**	**102452**	**102452.0**
frb59-26-4	**104445**	**104445.0**	**104445**	**104445.0**	**104445**	**104445.0**	**104445**	**104445.0**	**104445**	**104445.0**
frb59-26-5	**105533**	**105533.0**	**105533**	**105533.0**	**105533**	**105533.0**	**105533**	**105533.0**	**105533**	**105533.0**
socfb-Berkeley13	1173277	1173342.5	1173291	1173339.3	**1173249**	**1173318.9**	1173288	1173341.5	1173277	1173327.4
socfb-Indiana	**1599150**	**1599232.6**	1599217	1599299.1	1599184	1599258.3	1599186	1599262.9	1599162	1599275.9
socfb-OR	**2439133**	2439303.4	2439292	2439459.1	2439175	**2439257.7**	2439333	2439475.1	2439286	2439328.4
socfb-Penn94	2129692	**2129770.7**	**2129633**	2129844.2	2129692	2129785.9	2129722	2129849.3	2129753	2129829.4
socfb-UIllinois	**1639934**	**1640015.4**	1639937	1640063	1639942	1640067.3	1639978	1640063.4	1639980	1640043.9
BEST	8	8	6	5	6	7	5	5	5	5

superior performance of the proposed MAE-HTS algorithm in terms of both solution quality, computational efficiency, and robustness.

F. EXPERIMENTAL RESULTS ON ADDITIONAL LARGE-SCALE INSTANCES

In this experiment, we test our MAE-HTS algorithm on the instances obtained from the Network Data Repository, which are based on the real-world graphs. We compared our MAE-HTS algorithm with DLSWCC and NuMWVC which are obviously the best performing algorithms in the literature. These three algorithms were run under the same experimental conditions. Because of the large scale nature of the instances, the exact ILP solver Gurobi has difficulty in obtaining good solutions in a time limit of 10800s. For the 72 large-scale instances with different numbers of vertices and edges, all algorithms are executed for 10 times. The performance of all the three algorithms are reported in Table 7. Column *Best* reports the best objective values over 10 runs and columns *Avg* and *Times* show the average objective values and average computing time, respectively. The best values are indicated in bold too.

From Table 7, we can see that MAE-HTS and NuMWVC are better than DLSWCC on class ALPI. MAE-HTS obtains the best solutions on 64 instances while NuMWVC can find the best solutions on 27 instances. Moreover, MAE-HTS is more stable and robust than other reference algorithms with better average objective values, and it obtains the best average objective values on 64 instances of all 72 instances. Meanwhile, MAE-HTS achieves the best performance on class ALPI in a reasonable time (285.165s). In general, MAE-HTS demonstrates its effectiveness in solving these challenging large scale instances.

IV. ANALYSIS AND DISCUSSION

In this section, we conduct additional experiments to demonstrate the effectiveness of several important components in our MAE-HTS, including the master-apprentice evolutionary algorithm, the hybrid tabu search and the combination of the two searching strategies.

A. EFFECTIVENESS OF SEARCHING STRATEGIES IN MAE-HTS

The purpose of this experiment is to evaluate the effectiveness of the main components in MAE-HTS: the master-apprentice

evolutionary algorithm and the hybrid tabu search. Thus, we compare MAE-HTS with two variants of MAE-HTS: HTS and MAE, and the best reference algorithm NuMWVC. HTS is based on the hybrid tabu search and does not conduct the master-apprentice evolutionary procedure. MAE works without solution-based tabu search, i.e., it does not conduct the hash functions to record the visited solutions. In this experiment, the HTS and MAE are tested on 23 large scale representative instances from the class of ALPI instances over 10 runs. These instances include 10 instances from BHOSLIB and 13 instances from Facebook networks, which are difficult for the best reference algorithm NuMWVC in the literature.

As shown in Table 8, MAE-HTS outperforms the best performing algorithm NuMWVC as well as the two simplified versions of our MAE-HTS algorithm: HTS and MAE. First, HTS can obtain all the best solutions on 10 instances from BHOSLIB while MAE can only find the best solutions for 3 out of 10 instances. This disclose that the solution-based tabu search can progress a more precise search by using the memory of all the visited solutions to avoid cycling problem during the search. Second, for 13 instances from Facebook networks, compared to HTS, MAE performs better in terms of the best objective value and the average objective value over all 13 instances. Moreover, MAE is able to provide the best solutions on 1 instance. This demonstrates the importance of MAE especially for large scale instances, which increases the diversity and stability of the search process. Thus, the hybrid tabu search and the master-apprentice evolutionary algorithm are both important components in our MAE-HTS.

In fact, for most instances, HTS and MAE can also provide high quality solutions. Interested readers are referred to Table 10 in the Appendix for more details, which contains the results for HTS and MAE on all 72 instances of class ALPI.

B. PARAMETER SETTINGS

In our MAE-HTS, an important issue is to make a balance between local search and global search, i.e., how to decide the values of two important parameters *Iter_Cycle* and *Max_Iter*. In this experiment, we conduct experiments to fix the values of the two key parameters of the MAE-HTS algorithm.

- Parameter coefficient α for *Iter_Cycle*
- Parameter coefficient β for *Max_Iter*

The parameter coefficient represents the ratio between the new parameter in experiments and the original parameter. For example, $\alpha = 0.6$ means that the new parameter of *Iter_Cycle* is set as $5 \times 0.6 = 3$ while $\alpha = 1$ means the new parameter remains 5. We have chosen several representative instances as the testbed. These instances include 5 largest instances from BHOSLIB and 5 largest instances from Facebook networks. For this experiment, The algorithm was run for 10 times on each problem instance and each parameter setting with 1000s as the time limit. The results are presented in Table 9.

In Table 9, we notice that the algorithms with different settings can find the best solutions on both the best objective value and the average objective value for BHOSLIB instances. This indicates the stability of the MAE-HTS algorithm to some extent. However, we observe that the combination ($\alpha = 1, \beta = 1$) leads to better values for the 5 largest instances from Facebook networks. Thus, the parameters of

TABLE 10. Results on the class of ALPI by HTS and MAE.

Graph	HTS		MAE		MAE-HTS	
	Best	*Avg*	*Best*	*Avg*	*Best*	*Avg*
C2000-5	**136452**	136455.3	**136452**	136454.2	**136452**	**136452**
C2000-9	**139464**	**139464**	**139464**	**139464**	**139464**	**139464**
MANN-a45	**71923**	**71923**	**71923**	**71923**	**71923**	**71923**
MANN-a81	**234192**	**234192**	**234192**	**234192**	**234192**	**234192**
keller6	**230574**	**230598.6**	230608	230631.4	**230574**	230601.2
p-hat1500-1	**96959**	**96959**	**96959**	**96959**	**96959**	**96959**
p-hat1500-2	**100664**	**100664**	**100664**	**100664**	**100664**	**100664**
p-hat1500-3	**103593**	**103593**	**103593**	**103593**	**103593**	**103593**
frb56-25-1	**96479**	**96479**	96573	96575.9	**96479**	**96479**
frb56-25-2	**97136**	**97136**	**97136**	**97136**	**97136**	**97136**
frb56-25-3	**94638**	94644.6	94660	94660	**94638**	**94638**
frb56-25-4	**95476**	**95476**	**95476**	95669.3	**95476**	**95476**
frb56-25-5	**97978**	97982.8	98029	98033.2	**97978**	**97978**
frb59-26-1	**107857**	107917.2	108121	108121	**107857**	**107857**
frb59-26-2	**106154**	**106154**	106265	106288.4	**106154**	**106154**
frb59-26-3	**102452**	**102452**	**102452**	102494.6	**102452**	**102452**
frb59-26-4	**104445**	**104445**	104576	104623.5	**104445**	**104445**
frb59-26-5	**105533**	105625.6	105725	105725	**105533**	**105533**
bio-dmela	168722	168730	168704	168711.2	**168697**	**168704.7**
bio-yeast	**29557**	**29557**	**29557**	**29557**	**29557**	**29557**
ca-AstroPh	759257	759437.9	759198	**759262.9**	**759176**	759272
ca-citeseer	8192407	8192687.4	8192415	8192652.3	**8192170**	**8192470.4**
ca-CondMat	805922	806052.2	804905	**805012.8**	**804848**	805055.3
ca-CSphd	**33272**	**33272**	**33272**	**33272**	**33272**	**33272**
ca-dblp-2010	7727831	7728165.6	**7727693**	**7727945.7**	7728136	7728255.9
ca-dblp-2012	**10453866**	**10454043.9**	10453875	10454117.7	10454064	10454171.3
ca-Erdos992	**32550**	**32550**	**32550**	**32550**	**32550**	**32550**
ca-GrQc	139705	139716.1	139696	139696	**139694**	**139695.2**
ca-HepPh	431155	431360.8	430525	**430895.3**	**430518**	430899.3
ca-MathSciNet	8914783	8914869.1	8912834	**8912925.1**	**8912824**	8912960.7
ia-email-EU	**57326**	**57326**	**57326**	**57326**	**57326**	**57326**
ia-email-univ	**38443**	**38443**	**38443**	**38443**	**38443**	**38443**
ia-enron-large	810460	810644.5	809816	809929.5	**809620**	**809822.7**
ia-fb-messages	**37016**	**37016**	**37016**	**37016**	**37016**	**37016**
ia-reality	**5880**	**5880**	**5880**	**5880**	**5880**	**5880**
ia-wiki-Talk	1108805	1108821.5	1108780	1108790	**1108764**	**1108773.1**
rec-amazon	3037585	3038484.1	2995257	2995365	**2995015**	**2995183.3**
sc-nasasrb	3504816	3504920.7	**3504620**	**3504701.5**	3504644	3504777
sc-shipsec1	7878463	7879438	7876944	7877538.9	**7875579**	**7876523.5**
soc-brightkite	**1368681**	1368712.7	1368704	1368716.2	1368691	**1368702.7**
soc-delicious	5711658	5711847.3	5712341	5712915	**5711358**	**5711713.5**
soc-douban	**608796**	**608796**	**608796**	**608796**	**608796**	**608796**
soc-epinions	626096	626212.8	626072	626083.5	**626066**	**626075.7**
soc-gowalla	5450207	5500865.5	5449525	5490125.3	**5449285**	**5454981.6**
soc-slashdot	1433267	1433964.6	1433231	1433273	**1433158**	**1433201.6**
soc-twitter-follows	**161186**	**161186**	**161186**	**161186**	**161186**	**161186**
socfb-Berkeley13	1173323	1173465.9	**1173269**	**1173329.2**	1173277	1173342.5
socfb-CMU	340230	340258.8	340211	340224.1	**340209**	**340213.4**
socfb-Duke14	525195	525238.6	525160	525203.7	**525152**	**525169.7**
socfb-Indiana	1599408	1599575.9	1599187	1599269.8	**1599150**	**1599232.6**
socfb-MIT	318401	318424.6	318387	318394.6	**318385**	**318389.4**
socfb-OR	2439214	2439458.3	2439248	2439376.3	**2439133**	**2439303.4**
socfb-Penn94	2129897	2130057.5	2129724	2129811.4	**2129692**	**2129770.7**
socfb-Stanford3	586778	586821.8	586744	586762	**586742**	**586757.6**
socfb-UCLA	1034916	1035031.9	1034906	1034940	**1034894**	**1034924.5**
socfb-UConn	904141	904245.6	904005	**904072.2**	**904002**	904073.1
socfb-UCSB37	766490	766584.2	766325	766414.6	**766323**	**766391.8**
socfb-UIllinois	1640115	1640287.4	1639943	1640071.3	**1639934**	**1640015.4**
socfb-Wisconsin87	1258250	1258277	1258050	1258092.4	**1257975**	**1258031.3**
tech-internet-as	360079	360080.8	**360076**	360076.1	**360076**	**360076**
tech-p2p-gnutella	1085111	1085119.6	**1085103**	1085103.4	**1085103**	**1085103**
tech-RL-caida	4861135	4861518.7	4862181	4862773.2	**4860898**	**4861210.6**
tech-routers-rf	52117	52122.7	**52114**	**52114**	**52114**	**52114**
tech-WHOIS	147563	147571.8	**147559**	147560	**147559**	**147559.3**
web-arabic-2005	7679872	7680135.9	7678909	7679262.2	**7678576**	**7679167.6**
web-BerkStan	333010	333076	332839	332866.7	**332827**	**332841**
web-edu	90150	90172.3	90114	90115.9	**90108**	**90114**
web-google	**31942**	**31942**	**31942**	**31942**	**31942**	**31942**
web-indochina-2004	470043	470201.1	469912	**469955**	**469872**	469963
web-sk-2005	3655175	3655344.3	3654832	3655141.9	**3654734**	**3655057.7**
web-spam	150343	150358.3	150342	150350.7	**150341**	**150342.6**
web-webbase-2001	165785	165825.7	165706	165709.2	**165701**	**165704.4**
BEST	30	24	27	27	**67**	**61**

our algorithm are set as the initial value of *Iter_Cycle* and *Max_Iter*.

V. CONCLUSION

In this paper, we presented our MAE-HTS algorithm for solving the weighted vertex cover problem. By using multiple hash functions, MAE-HTS records the previously visited solutions effectively and thereby avoid cycling with high probability and intensifies the search within the search area of interest locally. By combining with a hybrid evolutionary algorithm, the diversification capability and the robustness of the MAE-HTS algorithm is enhanced significantly. The performance of MAE-HTS is evaluated and compared with the current best performing algorithms for the MWVCP on a set of public benchmark instances as well as additional large scale instances representing real-world graphs. The experimental results demonstrate the efficacy of the proposed MWVCP in terms of both solution quality and computational efficiency. Interestingly, we demonstrated that each component of the algorithm made a notable contribution to the success of our hybrid algorithm. However, we notice that NuMWVC has also achieved better results on some instances of ALPI. The reason might be that our MAE-HTS is realized by a two individual based evolutionary algorithm, a simple crossover operator and a greedy local search procedure, which has strong capability in local search but its capability of global search can be further enhanced. Thus, we can improve our HTS procedure by introducing random strategies or combining our HTS with other intelligence algorithms, such as monarch butterfly optimization (MBO), earthworm optimization algorithm (EWA), elephant herding optimization (EHO) and moth search (MS) algorithm, to enhance the global search capability of our algorithm in the future.

APPENDIX.

Table 10 presents the comparison results between HTS, MAE and MAE-HTS on 72 ALPI instances with 10 independent runs under the time limit of 1000 seconds for each run.

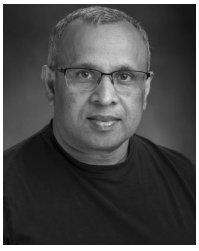
REFERENCES

[1] T. Bäck, D. B. Fogel, and Z. Michalewicz, *Handbook of Evolutionary Computation*. Boca Raton, FL, USA: CRC Press, 1997.
[2] S. R. Balachandar and K. Kannan, "A meta-heuristic algorithm for vertex covering problem based on gravity," *Int. J. Math. Stat. Sci.*, vol. 1, no. 3, pp. 130–136, 2009.
[3] S. Bouamama, C. Blum, and A. Boukerram, "A population-based iterated greedy algorithm for the minimum weight vertex cover problem," *Appl. Soft Comput.*, vol. 12, no. 6, pp. 1632–1639, Jun. 2012.
[4] J. Chen, I. A. Kanj, and G. Xia, "Improved parameterized upper bounds for vertex cover," in *Proc. Int. Conf. Math. Found. Comput. Sci.*, 2006, pp. 238–249.
[5] J. Ding, Z. Lü, C. M. Li, L. Shen, L. Xu, and F. Glover, "A two-individual based evolutionary algorithm for the flexible job shop scheduling problem," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 2262–2271.
[6] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. Berlin, Germany: Springer, 2003.
[7] T. A. Feo and M. G. C. Resende, "Greedy randomized adaptive search procedures," *J. Global Optim.*, vol. 6, no. 2, pp. 109–133, Mar. 1995.
[8] F. Glover, "Tabu search—Part I," *ORSA J. Comput.*, vol. 1, no. 3, pp. 190–206, 1989.
[9] F. Glover, "Tabu search—Part II," *ORSA J. Comput.*, vol. 2, no. 1, pp. 4–32, Feb. 1990.
[10] F. Glover and M. Laguna, "Tabu search," in *Handbook of Combinatorial Optimization*. Boston, MA, USA: Springer, 1998, pp. 2093–2229.
[11] G. Gong, Q. Deng, R. Chiong, X. Gong, and H. Huang, "An effective memetic algorithm for multi-objective job-shop scheduling," *Knowl.-Based Syst.*, vol. 182, Oct. 2019, Art. no. 104840.
[12] J. K. Hao, "Memetic algorithms in discrete optimization," in *Handbook of Memetic Algorithms*. Berlin, Germany: Springer, 2012, pp. 73–94.
[13] J. H. Holland, "Adaptation in natural and artificial systems," *Quart. Rev. Biol.*, vol. 6, no. 2, pp. 126–137, 1975.
[14] R. Jovanovic and M. Tuba, "An ant colony optimization algorithm with improved pheromone correction strategy for the minimum weight vertex cover problem," *Appl. Soft Comput.*, vol. 11, no. 8, pp. 5360–5366, Dec. 2011.
[15] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*. Boston, MA, USA: Springer, 1972, pp. 85–103.
[16] N. Krasnogor and J. Smith, "A tutorial for competent memetic algorithms: Model, taxonomy, and design issues," *IEEE Trans. Evol. Comput.*, vol. 9, no. 5, pp. 474–488, Oct. 2005.
[17] C. M. Li, H. Jiang, and R. C. Xu, "Incremental MaxSAT reasoning to reduce branches in a branch-and-bound algorithm for MaxClique," in *Proc. Int. Conf. Learn. Intell. Optim.* Cham, Switzerland: Springer, 2015, pp. 268–274.
[18] R. Li, S. Hu, S. Cai, J. Gao, and M. Yin, "NuMWVC: A novel local search for minimum weighted vertex cover problem," *J. Oper. Res. Soc.*, vol. 71, pp. 1498–1509, Sep. 2020.
[19] R. Li, S. Hu, H. Zhang, and M. Yin, "An efficient local search framework for the minimum weighted vertex cover problem," *Inf. Sci.*, vol. 372, pp. 428–445, Dec. 2016.
[20] Z. Lü, F. Glover, and J.-K. Hao, "A hybrid Metaheuristic approach to solving the UBQP problem," *Eur. J. Oper. Res.*, vol. 207, no. 3, pp. 1254–1262, Dec. 2010.
[21] Z. Lü and J.-K. Hao, "A memetic algorithm for graph coloring," *Eur. J. Oper. Res.*, vol. 203, no. 1, pp. 241–250, May 2010.
[22] L. Moalic and A. Gondran, "Variations on memetic algorithms for graph coloring problems," *J. Heuristics*, vol. 24, no. 1, pp. 1–24, Feb. 2018.
[23] P. Moscato and C. Cotta, "A gentle introduction to memetic algorithms," in *Handbook of Metaheuristics*. Boston, MA, USA: Springer, 2003, pp. 105–144.
[24] G. L. Nemhauser and L. E. Trotter, "Vertex packings: Structural properties and algorithms," *Math. Program.*, vol. 8, no. 1, pp. 232–248, Dec. 1975.
[25] R. Niedermeier and P. Rossmanith, "On efficient fixed-parameter algorithms for weighted vertex cover," *J. Algorithms*, vol. 47, no. 2, pp. 63–77, Jul. 2003.
[26] P. Pandey and A. P. Punnen, "The generalized vertex cover problem and some variations," *Discrete Optim.*, vol. 30, pp. 121–143, Nov. 2018.
[27] M. G. Resende and C. C. Ribeiro, "Greedy randomized adaptive search procedures: Advances, hybridizations, and applications," in *Handbook of Metaheuristics*. Boston, MA, USA: Springer, 2010, pp. 283–319.
[28] R. Rossi and N. Ahmed, "The network data repository with interactive graph analytics and visualization," in *Proc. 29th AAAI Conf. Artif. Intell.*, 2015, pp. 31–49.
[29] W.-T. Shiue, "Novel state minimization and state assignment in finite state machine design for low-power portable devices," *Integr. VLSI J.*, vol. 38, no. 4, pp. 549–570, Apr. 2005.
[30] S. J. Shyu, P.-Y. Yin, and B. M. T. Lin, "An ant colony optimization algorithm for the minimum weight vertex cover problem," *Ann. Oper. Res.*, vol. 131, nos. 1–4, pp. 283–304, Oct. 2004.
[31] A. Singh and A. K. Gupta, "A hybrid heuristic for the minimum weight vertex cover problem," *Asia–Pacific J. Oper. Res.*, vol. 23, no. 2, pp. 273–285, Jun. 2006.
[32] W. Sun, J.-K. Hao, W. Wang, and Q. Wu, "Memetic search for the equitable coloring problem," *Knowl.-Based Syst.*, vol. 188, Jan. 2020, Art. no. 105000.
[33] C. Tang, A. Li, and X. Li, "Asymmetric game: A silver bullet to weighted vertex cover of networks," *IEEE Trans. Cybern.*, vol. 48, no. 10, pp. 2994–3005, Oct. 2018.
[34] L. Wang, W. Du, Z. Zhang, and X. Zhang, "A ptas for minimum weighted connected vertex cover P_3 problem in 3-dimensional wireless sensor networks," *J. Combinat. Optim.*, vol. 33, no. 1, pp. 106–122. 2017.

[35] L. Wang, C.-M. Li, J. Zhou, B. Jin, and M. Yin, "An exact algorithm for minimum weight vertex cover problem in large graphs," 2019, *arXiv:1903.05948*. [Online]. Available: http://arxiv.org/abs/1903.05948
[36] H. Xu, T. S. Kumar, and S. Koenig, "A new solver for the minimum weighted vertex cover problem," in *Proc. Int. Conf. AI OR Techn. Constraint Program. Combinat. Optim. Problem*. Cham, Switzerland: Springer, 2016, pp. 392–405.
[37] X. Xu and J. Ma, "An efficient simulated annealing algorithm for the minimum vertex cover problem," *Neurocomputing*, vol. 69, nos. 7–9, pp. 913–916, Mar. 2006.
[38] T. Zhou, Z. Lü, Y. Wang, J. Ding, and B. Peng, "Multi-start iterated tabu search for the minimum weight vertex cover problem," *J. Combinat. Optim.*, vol. 32, no. 2, pp. 368–384, Aug. 2016.

YANG WANG was born in Hubei, China, in 1994. He received the B.S. degree in computer science from the Huazhong University of Science and Technology, Wuhan, China, in 2015, where he is currently pursuing the Ph.D. degree in computer software and theory. His research interests include combinatorial optimization, global optimization, and heuristic search algorithms.

ZHIPENG LÜ received the B.S. degree in applied mathematics from Jilin University, China, in 2001, and the Ph.D. degree in computer software and theory from the Huazhong University of Science and Technology, China, in 2007. From 2007 to 2011, he was a Postdoctoral Research Fellow with LERIA, Department of Computer Science, University of Angers, France. He is currently a Professor with the School of Computer Science and Technology, Huazhong University of Science and Technology, and the Director of the Institute of Artificial Intelligence and Optimization. His research interests include artificial intelligence, computational intelligence, operations research, and adaptive metaheuristics for solving large scale real-world and theoretical combinatorial optimization and constrained satisfaction problems.



ABRAHAM P. PUNNEN received the Ph.D. degree in operations research from IIT Kanpur, in 1990. Prior to joining SFU, he was a Professor with the University of New Brunswick, Saint John, Canada. He held visiting appointments at University Colorado, Denver, USA, University of Kentucky, Lexington, USA, and Universite Catholique de Louvain, Belgium, under the prestigious CORE Fellowship. He is currently a Professor with the Department of Mathematics, Simon Fraser University, Surrey, BC, Canada, and the Founding Director of the Center for Operations Research and Decision Sciences, SFU. He published over 100 articles in major research journals and books, and is well known for the book he edited jointly with G. Gutin on the traveling salesman problem. He holds various research grants from industry and government. He has supervised many graduate students and postdoctoral fellows. He is also a member of the NSERC Grant Evaluation Group in Civil, Environmental, and Industrial Engineering.