A novel local search for minimum weighted vertex cover problem

Paper #202

Abstract

The problem of finding a minimum weighted vertex cover (MWVC) in a graph is a well known combinatorial optimization problem with important applications. This paper introduces a novel local search algorithm called SARRCCP for MWVC based on three heuristics. Firstly, four reduction rules are introduced during the construction phase. Secondly, a new variant of configuration checking strategy, as called configuration checking with priority criterion strategy, is proposed for reducing cycling problem during the search. Moreover, a self-adaptive vertex removing strategy is proposed to balance the intensification search and diversification search. Experimental results show that SARRCCP outperforms state-of-art local search algorithms for MWVC on both the standard benchmarks and real world massive graphs benchmarks.

1 Introduction

Given an undirected graph $G = (V, E)$, a vertex cover C of G is a subset of V such that for each $e \in E$, at least one of e's endpoints is in C. The minimum vertex cover problem (MVC) consists in finding a vertex cover with minimum number of vertices. An important generalization of MVC is the minimum weighted vertex cover problem (MWVC), in which each vertex is associated with a non-negative integer, and the goal is to find a vertex cover with the smallest total weight. Obviously, MWVC reduces to MVC if each vertex is associated with the same weight. The MWVC problem is an important combinatorial problem and has been widely used in various fields, such as wireless communication, circuit design and network flows.

As is well known, the decision version of MVC is one of the prominent Karp's 21 NP-complete combinatorial problems [Karp, 1972]. Moreover, it is NP-hard to approximate MVC within any factor smaller than 1.3606 [Dinur and Safra, 2005]. Therefore, for large and hard instances, researchers usually resort to heuristic approaches to obtain good solutions within reasonable time. In the past decade, several heuristic algorithms have been proposed to solve MVC, e.g., COVER [Richter *et al.*, 2007], EWLS [Cai *et al.*, 2010], EWCC[Cai *et al.*, 2011], NuMVC [Cai *et al.*, 2013], TwMVC [Cai *et al.*, 2015], FastVC [Cai, 2015], and Lincom [Fan *et al.*, 2015]. Additionally, the MVC problem is closely related to the maximum clique problem (MCP), and algorithms for those two problems can be directly used to solve MVC as well.

For the MWVC problem, numerous research efforts have also been made on the development of heuristics. [Balaji *et al.*, 2010] proposes an effective algorithm, as called Support Ratio Algorithm (SRA). [Voß and Fink, 2012] proposes a modified reactive tabu search. In [Balachandar and Kannan, 2009], a new heuristic operator is introduced in the domain of randomized gravitational emulation search algorithm (RGES). In addition, [Shyu *et al.*, 2004] introduces a meta-heuristic based on the ant colony optimization (ACO) approach. Afterwards, [Jovanovic and Tuba, 2011] improves this algorithm. [Bouamama *et al.*, 2012] proposes a population-based iterated greedy algorithm. Especially, a recent algorithm called MS-ITS [Zhou *et al.*, 2015], which clearly dominates other local search algorithms, makes a significant improvement in MWVC solving.

On the other hand, recent advances in internet have resulted in a collection of massive graph data sets consisting of millions of vertices. These graphs are larger than what most previous algorithms have been designed for, and therefore call for new heuristics and algorithms. [Cai, 2015] focuses on solving the MVC on massive graphs and designs a local search that makes a balance between the time efficiency and the guidance effectiveness of heuristic. Based on that, [Fan *et al.*, 2015] uses several reduction rules during the construction stage of the local search, and proposes a new data structure to improve the time efficiency. For maximum weighted clique problem (MWCP), [Wang *et al.*, 2016] proposes a low time-complexity heuristic as called BPS to improve the local search. It should be pointed out here that, though MWVC has close connection with MWCP, for most of the massive graphs, which are very sparse, solving MWVC by MWCP on the complementary graphs would suffer from the problem that the complementary graphs grow too large to handle [Fang *et al.*, 2014; Li *et al.*, 2015].

The aim of this paper is to design a novel local search algorithm for MWVC. To achieve good performance on both standard benchmarks and massive graph benchmarks, several techniques are adopted. Firstly, the reduction rules have been proved to be efficient when handling massive graphs. For example, as is mentioned in [San Segundo *et al.*, 2016],

the well known *k*-core structure can be used to solve maximum clique problem to reduce the instance size. This structure can be easily adapted to solve MWCP effectively. For the MVC problem, reduction rules have also been used by [Fan *et al.*, 2015], and experimental results show that the reduction heuristic itself can obtain better results compared with previous algorithms. Following this line of research, we introduce four reduction rules to obtain a better initial solution for subsequent search process. These rules can be viewed as a weighted version for those rules used in [Fan *et al.*, 2015], and are first used to design a local search for MWVC.

The second technique we adopt in this paper is a new variant of the configuration checking (CC) strategy. The CC strategy is first designed by [Cai *et al.*, 2011] for tackling the cycling problem for local search, and has gained great success in solving different kinds of combinatorial problems, such as MVC [Cai *et al.*, 2011], SCP [Wang *et al.*, 2015], SAT [Cai and Su, 2012; 2013; Luo *et al.*, 2015], MaxSAT [Luo *et al.*, 2014], and MWCP [Wang *et al.*, 2016]. But a direct application of CC strategy cannot result in an efficient MWVC algorithm, because the forbidding strength of original CC is too strong in the context of MWVC. In other words, CC forbids any vertex whose circumstance has not changed since its last state was changed, regardless of the benefit its state changing can bring. It is interesting to point out here that in [Wang *et al.*, 2016], a strong version of CC strategy has been proposed because original CC is too weak when solving MWCP. Conversely, we propose a weak version of CC strategy as called configuration checking with priority criterion strategy (CCP for short) for MWVC. The CCP strategy is looser than original CC strategy, and therefore can improve the exploiting ability of the local search.

The third idea, namely self-adaptive vertex removing strategy, is proposed to improve the local search process. As we know, in previous local search scheme for MWVC and MVC, once a *k*-sized vertex cover is found, one vertex would be removed from the candidate solution, and the algorithm then starts from the current vertex set to search for a (k-1)-sized vertex cover. In this paper, we will not fix the number of the deleted vertex to be one. Specifically, we propose a self-adaptive heuristic to decide the number of the removed vertex in each iteration.

By incorporating the above three heuristics, we propose a novel local search framework for MWVC, as called SARRCCP (Self-adaptive local search based on reduction rules and configuration checking with priority criterion). Experimental results show that our algorithm shows its superiority on both standard benchmarks and massive graph benchmarks compared with state-of-art local search algorithms. We also conduct experiments to analyze the effectiveness of the proposed three heuristics.

In the next section, we introduce some necessary background knowledge. After that, we propose the reduction rules for constructing starting weighted vertex covers. Next we describe the configuration checking with priority criterion. Then, we present the self-adaptive heuristic and the SARRCCP algorithm. Experimental results demonstrating the performance of SARRCCP are presented next. Finally we give some concluding remarks.

2 Preliminaries

An undirected weighted graph $G = (V, E, w)$ consists of a vertex set V and an edge set E, where $V = \{v_1, v_2, \ldots, v_n\}$, $E = \{e_1, e_2, \ldots, e_m\}$ and each vertex $v_i (i = 1, 2, \ldots, n)$ is associated with a weight $w(v_i)$. Each edge is a 2-element subset of V. For an edge $e = \{u, v\}$, we say that vertices u and v are the endpoints of edge e. Two vertices are neighbors if and only if they both belong to some edge. The neighborhood of a vertex v is denoted as $N(v) = \{u \in V | \{u, v\} \in E\}$.

For an undirected weighted graph $G = (V, E, w)$, a vertex cover is a subset of V which contains at least one of the two endpoints of each edge. We are concerned in this paper with the problem of finding a vertex cover with the minimum sum of weights of the belonging vertices. An edge $e \in E$ is covered by a candidate solution C if at least one endpoint of e belongs to C, and otherwise it is uncovered. The age of a vertex is defined as the number of search steps that have occurred since its state was last changed.

Usually, MWVC local search algorithms maintain a current candidate solution during the search. For convenience, in the rest of this paper, we use C to denote the current candidate solution. The state of a vertex v is denoted by $s_v \in \{1, 0\}$, such that $s_v = 1$ means $v \in C$, and $s_v = 0$ means $v \notin C$. In our local search algorithm for MWVC, each edge $e \in E$ is associated with a non-negative integer number $edge_w(e)$ as its edge weight. The $edge_w$ of each edge is maintain as follow. In the initialization process, for each edge $e \in E$, $edge_w(e)$ is set to 1. In the end of each loop during the local search process, each edge $e \in E$ will be checked whether e is covered by the candidate solution. If e is uncovered, $edge_w(e)$ will be increased by 1. Given a candidate solution C, the cost of C, denoted by $cost(C)$, is defined as the total weight of edges uncovered by C. MWVC local search algorithms search for candidate solutions with lower cost. For a vertex v, $score(v) = (cost(C) - cost(C'))/w(v)$ where $C' = C \backslash \{v\}$if $v \in C$, and $C = C \cup \{v\}$ otherwise, measuring the benefit of changing the state of vertex v.

3 The Construction Stage

In this section, we shall introduce four reduction rules and show how these rules can be used to construct an initial candidate solution for subsequent local search.

3.1 Reduction Rules

The following reduction rules shall be used in the constructing procedure to handle vertices of small degrees.

Weighted-Degree-1: If G contains a vertex u s.t. $N(u) = \{v\}$ and $w(u) \geq w(v)$, then there is a minimum weighted vertex cover of G that contains v.

Weighted-Degree-2 with Triangle-1 Rule: If G contains a vertex v s.t. $N(v) = \{n_1, n_2\}$, $\{n_1, n_2\} \in E$ and $w(v) \geq w(n_1) + w(n_2)$, then there is a minimum weighted vertex cover of G that contains both n_1 and n_2.

Weighted-Degree-2 with Triangle-2 Rule: If G contains a vertex v s.t. $N(v) = \{n_1, n_2\}$, $N(n_1) = \{v, n_2\}$, $\{n_1, n_2\} \in E$ and $w(v) \geq w(n_1)$, then there is a minimum weighted vertex cover of G that contains n_1.

Weighted-Degree-2 with Quadrilateral Rule: If G contains two vertices u and v s.t. $N(u) = N(v) = \{n_1, n_2\}$, $\{n_1, n_2\} \notin E$ and $w(u) + w(v) \geq w(n_1) + w(n_2)$, then there is a minimum weighted vertex cover of G that contains both n_1 and n_2.

Although reduction rules for the MWVC are inspired by the reduction rules for the MVC used in Lincom [Fan *et al.*, 2015], they still have some differences. Firstly, in Lincom, the reduction rules do not need to consider the weight of each vertex, while reduction rules for the MWVC need to consider the weight of each vertex. So in each reduction rule, a weighted constraint is added. Secondly, another situation of Degree-2 with Triangle Rule, i.e. **Weighted-Degree-2 with Triangle-2 Rule** is incorporated. This rule is from a theoretical paper [Niedermeier and Rossmanith, 2003], and is never applied to design a local search algorithm so far as we know.

3.2 Constructing a weighted vertex cover with reductions

The first step of our local search algorithm is to propose a construction procedure, which is outlined in Algorithm 1. Inspired by [Cai, 2015], the construction procedure of our algorithm consists of an extending phase (Lines 2 to 9) and a shrinking phase (Line 10). In Line 1, C is initialized to be an empty set. From Lines 3 to 6, the proposed reduction rules are applied to put vertices which must be in the optimal solution into C until there is no rule satisfied. After that, in Lines 8 and 9, the procedure extends C to be a weighted vertex cover of G, by checking and covering each edge in each iteration: if the considered edge is uncovered, the endpoint with a higher $score(v)$ is added into C. In this way, an weighted vertex cover can be obtained at the end of the extending phase. In the shrinking phase (Line 9), the redundant vertices are removed from C similar to what [Cai 2015] did.

Algorithm 1: Construct WVC( )

Input: A graph $G = (V, E, w)$
Output: An initial minimum vertex cover C of G

```
1  C = ∅;
2  while any rule is satisfied do
3      Repeatedly execute Weighted-Degree-1 Rule until it is
       not satisfied;
4      Repeatedly execute Weighted-Degree-2 with Triangle-1
       Rule until it is not satisfied;
5      Repeatedly execute Weighted-Degree-2 with Triangle-2
       Rule until it is not satisfied;
6      Repeatedly execute Weighted-Degree-2 with
       Quadrilateral Rule until it is not satisfied;
7  for e ∈ E do
8      if e is uncovered then
9          add the endpoint of e with higher score into C;
10 C ← RemoveRedundantVertices();
11 return C*;
```

After the construction procedure, if a vertex v is put into C based on the four reduction rules, there exists a minimum weighted vertex cover containing v. We call such kind of vertex v an *inferred weighted* vertex. During the subsequent local search procedure, these *inferred weighted* vertices are not allowed to be removed from the candidate solution.

4 Configuration Checking with priority Criterion

The configuration checking (CC) strategy is first proposed in [Cai *et al.*, 2011] to reduce cycling problem during local search for MVC. The idea of CC can be described as follows. Given a vertex v, its configuration denotes the states of all its neighboring vertices. If v's configuration remains the same as the last time it was removed from the candidate set, then it is forbidden to be added back into the candidate set. The CC strategy is usually implemented with a Boolean array named $config$, where $config(v) = 1$ means v is allowed to be added into the candidate solution and $config(v) = 0$ means v is forbidden to be added into the candidate solution.

The CC strategy for MVC can be modified for MWVC straightforwardly. In the beginning, the value of $config(v)$ is set to be 1 for each vertex. Afterwards, if a vertex v is removed from C, the value of $config(v)$ is set to be 0 and for each vertex v' of its neighbourhood, $config(v')$ = 1. Similarly, if a vertex v is added into C, for each vertex v' of its neighbourhood, $config(v)$ = 1. Unfortunately, such kind of a direct adaption of CC strategy cannot result in an efficient algorithm. A careful analysis shows that the original CC strategy is sometimes so strict that it would mislead the search by forbidding some promising candidate vertices.

For the add operation, original CC strategy forbids any vertex to be added into the candidate solution if its configuration has not been changed. However, in MWVC, it is quite reasonable to classify the vertices whose $config$ values equal 0 into two classes. For a vertex v in the first class, adding v into the current candidate solution can result in a better weighted vertex cover. Conversely, adding a vertex v' in the second class into the current candidate solution cannot result in a better weighted vertex cover. Indeed, the vertices in the first class are encouraged to be added into the weighted vertex cover.

For the remove operation, a vertex is removed to help the search jump out of local optimum. In this case, an *inferred weighted* vertex introduced in section 3 could not be removed since it must be in the minimum weighted vertex cover according to the reduction rules.

Based on above consideration, a new variant of configuration checking strategy, as we call configuration checking with priority criterion (CCP for short), is proposed. Specifically, a priority criterion is a condition which allows a vertex unsatisfying the CC criterion to be added into C only if it can result in a better solution compared with the current best-known solution. During the local search, the $config$ array is maintained as follows.

CCP-Rule 1: In the beginning, for each vertex v, $config(v)$ is initialized as 1.

CCP-Rule 2: When removing v from C, $config(v)$ is reset to 0, for each $u \in N(v)$, $config(u)$ is set to 1.

CCP-Rule 3: When adding v into C, for each $u \in N(v)$, $config(u)$ is set to 1.

Based on CCP strategy, the vertex selecting strategy is defined as follows.

Remove-Rule: Removing one vertex v, which has the highest $score(v)$ value, and is not an *inferred weighted* vertex, breaking ties in favor of the oldest one, and then updating the CCP values of v and its neighbors.

Add-Rule: Adding one vertex v with highest $score(v)$ value, regardless of its $config$ value, if v satisfies the priority criterion, and breaking ties in favor of the oldest one; otherwise, adding one vertex v with highest $score(v)$ value, whose $config$ value equals 1, and breaking ties in favor of the oldest one. Then updating the $config$ values of its neighbors.

In a nut shell, the original CC strategy only allows a vertex v to be added into the current vertex cover when the state of some of v's neighbourhoods have been changed, while CCP strategy allows the adding of v regardless of its CCP value when the added vertex can improve the current best solution. Therefore, the CCP strategy is greedier than CC and thus can help to lead the search to a more promising space.

5 Self-adaptive strategy

A general framework of previous local search algorithm for MVC is shown in Algorithm 2. As can be seen in this framework, once a k-sized vertex cover is found, the algorithm removes one vertex from C and focus on searching for a $(k-1)$-sized vertex cover. In this way, the local search iteratively finds a better solution step by step.

Algorithm 2: Local Search with step-by-step vertex removing strategy( )

```
1 (C, optInfo) ← InitVC();
2 while not reach terminate condition do
3     if C covers all edges then
4         C* ← C;
5         remove one vertex from C;
6     exchange a pair of vertices;
7 return C*;
```

In this section, we discuss the drawback of the step-by-step vertex removing strategy. We can observe that, in the early stage of the local search process, the candidate solution is far from the optimal solution, so the intensification search is more important this time. Therefore the local search algorithm should make forward by a big step rather than step by step to improve the current solution. While in the late stage of the local search process, the candidate solution is close to the optimal solution, and the diversification search is more important this time. Therefore the local search algorithm should make forward by a smaller step.

Base on above consideration, a self-adaptive vertex removing strategy (SAVR) is proposed to make a balance between intensification search and diversification search. The details of the strategy can be viewed in Algorithm 3. At first, the local search removes *remove-num* vertices for each iteration. With the increasing of iterations, if the algorithm can not find a better solution for β times, the number of *remove-num* would be decreased by 1. In our experiments, the value of *remove-num* is set to be 3, and β is set to be 50.

Algorithm 3: Self-adaptive-vertex-removing( )

```
1 if No-improve == β and remove-num≠ 1 then
2     remove-num−−;
3 for i = 0; i <remove-num; i + + do
4     remove v according to Remove-Rule;
```

6 The SARRCCP algorithm

Based on the techniques discussed above, we develop a local search algorithm for MWVC named SARRCCP. The pseudo code of the algorithm is shown in Algorithm 4. Generally speaking, the algorithm consists of two stages: the construction stage (Line 1) and the local search stage (Line 4 to 12). In the beginning, an initial weighted vertex cover C is constructed by the $ConstructWVC$ procedure as we discussed in section 3.2.

After the initialization process, the main outer loop from Line 4 to Line 12 is executed until reaching time limit. At first, the algorithm selects several vertices to be removed from the candidate solution (Line 5), according to the self-adaptive vertex removing strategy introduced in section 5. Additionally, note that these removed vertices must obey the **Remove-Rule** discussed in Section 4.

After the self-adaptive removing process, SARRCCP iteratively adds a vertex into C according to the **Add-Rule** until the candidate solution covers all edges, and then the redundant vertices are removed from C (Line 6-8). If a better solution is obtained, i.e. C is smaller than the current best solution C^*, C^* is updated by C (Line10) and the value of *No-improve* is reset to be 0; otherwise, *No-improve*++. When stop criterion is satisfied, the best solution will be returned.

Algorithm 4: SARRCCP( )

```
   Input: A graph G = (V, E, w)
   Output: A minimum vertex cover C of G
1  C ← ConstructWVC();
2  C* ← C;
3  No-improve= 0;
4  while stop criterion is not satisfied do
5      Self-adaptive-vertex-removing( );
6      while C uncovers some edges do
7          add v according to Add-Rule;
8      C ← RemoveRedundantVertices();
9      if C is smaller than C* then
10         C* ← C;
11         No-improve= 0;
12     else No-improve++;
13 return C*;
```

7 Experimental evaluation

In this section, we carry out extensive experiments to evaluate the performance of the SARRCCP algorithm for MWVC on four benchmarks, LPI [Shyu *et al.*, 2004], BHOSLIB [Xu *et al.*, 2007], DIMACS [Johnson and Trick, 1996] and real-world massive graphs [Rossi and Ahmed, 2015]. For sake

Table 1: Experiment results of MS-ITS and SARRCCP on BHOSLIB and DIMACS.

Instance	MS-ITS		SARRCCP		$\delta_{Best}(\delta_{Avg})$
	Best(Avg)	time(s)	Best(Avg)	time(s)	
500*500	12623(12635)	2.7	**12616(12616)**	0.4	7(19)
500*1000	16480(16483.1)	3.9	**16465(16465)**	1.1	15(18.1)
500*2000	**20863**(20866.9)	1.1	**20863(20863)**	1.2	0(3.9)
500*5000	**27241(27241)**	1.5	**27241(27241)**	1.4	0(0)
500*10000	**29573(29573)**	0.8	**29573(29573)**	0.6	0(0)
800*500	15046(15054.1)	3.9	**15025(15025)**	0.8	21(29.1)
800*1000	22760(22760)	4.9	**22747(22747)**	0.6	13(13)
800*2000	31309(31345.7)	7.0	**31287(31301.7)**	9.0	22(44)
800*5000	**38553(38557.1)**	6.0	**38553**(38566)	9.4	0(-8.9)
800*10000	**44351**(44359.9)	3.2	**44351(44353)**	4.4	0(6.9)
1000*1000	24735(24766.1)	10.4	**24723(24723)**	0.7	12(43.1)
1000*5000	45230(45256.9)	6.6	**45215(45238.9)**	6.5	15(18)
1000*10000	**51378**(51423)	4.0	**51378(51387.1)**	8.4	0(35.9)
1000*15000	58014(58068.9)	4.1	**57994(57994.5)**	7.0	20(74.4)
1000*20000	59675(59719.9)	4.6	**59651(59669.5)**	9.9	24(50.4)
C2000.5	**119598(119598.0)**	5248.2	**119598**(119620.9)	179.7	0.0(-22.9)
C2000.9	114645(114695.8)	1814.8	**114515(114582.4)**	33.4	130.0(113.4)
MANN-a27	13300(13303.4)	2.9	**13252(13252.0)**	0.8	48.0(51.4)
MANN-a45	35132(35147.2)	0.7	**35090(35090.0)**	0.9	42.0(57.2)
MANN-a81	**109068(109068.0)**	3.2	**109068(109068.0)**	0.0	0.0(0.0)
keller6	196830(196858.0)	13161.8	**196596(196667.9)**	106.5	234.0(190.1)
p-hat1500-1	**88694(88694.0)**	83.8	**88694(88694.0)**	2.6	0.0(0.0)
p-hat1500-2	**85251(85251.0)**	26.9	**85251(85251.0)**	3.0	0.0(0.0)
p-hat1500-3	**83449(83449.0)**	16.7	**83449(83449.0)**	2.0	0.0(0.0)
frb56-25-1	**78944(78944.0)**	439.4	**78944(78944.0)**	9.6	0.0(0.0)
frb56-25-2	78435(78462.2)	507.9	**78422(78430.8)**	23.0	13.0(31.4)
frb56-25-3	79636(79663.2)	913.3	**79600(79621.6)**	13.0	36.0(41.6)
frb56-25-4	79631(79653.2)	645.1	**79611(79627.7)**	17.9	20.0(25.5)
frb56-25-5	79913(79919.0)	629.2	**79906(79915.0)**	19.3	7.0(4.0)
frb59-26-1	86531(86538.0)	1238.4	**86504(86512.1)**	15.8	27.0(25.9)
frb59-26-2	**87759**(87838.6)	786.6	**87759(87766.9)**	15.6	0.0(71.7)
frb59-26-3	88677(88705.8)	1099.3	**88642(88679.0)**	23.0	35.0(26.8)
frb59-26-4	**87136**(87161.6)	1059.3	**87136(87151.6)**	21.8	0.0(10.0)
frb59-26-5	87883(87935.4)	663.9	**87871(87892.1)**	19.7	12.0(43.3)

of space, we only report LPI instances with more than 500 vertices and omit the SPI and MPI instances whose vertices numbers are all less than 300. For the same reason, DIMACS instances that can be solved easily by all the compared algorithms are not reported, and three groups of the largest-sized BHOSLIB instances are selected. We translate BHOSLIB, DIMACS, and massive graphs to MWVC instances by using the same method reported in [Shyu *et al.*, 2004]. The vertex weights are randomly drawn from a uniform distribution from the interval [20, 120].

For comparison, we choose MS-ITS [Zhou *et al.*, 2015] to represent a state-of-the-art algorithm for solving MWVC. Although MS-ITS algorithm outperforms other local search algorithm for MWVC, it seems that it fails to find a weighted vertex cover for many of the massive graphs. For further comparison, we rewrite the code of FastVC and obtain FastWVC algorithm which can solve MWVC. FastWVC shares the same construction phase and Best from multiple heuristic as FastVC. The main difference between these algorithms is that FastWVC use the same scoring function to evaluate each vertex as that of SARRCCP.

SARRCCP and FastWVC are both implemented in C++, and are compiled by g++ 4.6.2 with the -O2 option. The source code of MS-ITS is offered by the authors. All the algorithms are executed on a computer with Intel(R) Xeon(R) CPU E7-4830 with 2.13GHz. For each instance, each algorithm is performed 20 independent runs with different random seeds, where each run is terminated upon reaching a given time limit (1000 seconds).

For each algorithm on each instance, we report the minimum solution values Best and average solution values Avg obtained by the algorithms. δ_{Best} and δ_{Avg} indicate the dif-

Table 2: Experiment results of MS-ITS, FastWVC, and SARRCCP on the massive graphs.

Instance	MS-ITS	FastWVC	SARRCCP	$\delta_{Best}(\delta_{Avg})$
	Best(Avg)	Best(Avg)	Best(Avg)	
bio-dmela	149452(149556.8)	149371(149416.6)	**148499(148502.4)**	872(914.2)
bio-yeast	24269(24290.0)	24412(24427.6)	**24265(24265.0)**	147(162.6)
ca-AstroPh	662655(662926.5)	647677(647885.6)	**645000(645070.6)**	2677(2815.0)
ca-citeseer	N/A(N/A)	7035308(7035742.6)	**7029398(7031461.2)**	5910(4281.4)
ca-coauthors-dblp	N/A(N/A)	**27097178(27098041.0)**	27113616(27124354.3)	-16438(-26313.3)
ca-CondMat	704287(704798.5)	686568(686877.3)	**683659(683745.7)**	2909(3131.6)
ca-CSphd	29550(29609.8)	29452(29490.8)	**29390(29390.0)**	62(100.8)
ca-dblp-2010	N/A(N/A)	6606927(6607233.4)	**6600741(6601770.0)**	6186(5463.4)
ca-dblp-2012	N/A(N/A)	8971796(8972443.2)	**8962345(8963590.6)**	9451(8852.6)
ca-Erdos992	28303(28303.0)	**28298(28298.0)**	**28298(28298.0)**	0(0.0)
ca-GrQc	122330(122331.5)	122749(122797.2)	**122250(122254.3)**	499(542.9)
ca-HepPh	372836(373069.8)	365348(365564.1)	**363976(364001.4)**	1372(1562.7)
ca-hollywood-2009	N/A(N/A)	**48932617(48934884.2)**	48969670(48973837.1)	-37053(-38952.9)
ca-MathSciNet	N/A(N/A)	7646167(7648798.8)	**7637056(7637594.9)**	9111(11203.9)
ia-email-EU	N/A(N/A)	48322(48348.1)	**48269(48269.0)**	53(79.1)
ia-email-univ	**32931**(32933.0)	32998(33005.9)	**32931(32931.0)**	67(74.9)
ia-enron-large	N/A(N/A)	695227(695481.4)	**691571(691651.7)**	3656(3829.7)
ia-fb-messages	**32300**(32316.5)	32336(32338.6)	**32300(32300.0)**	36(38.6)
ia-reality	**4894(4894.0)**	**4894(4894.0)**	**4894(4894.0)**	0(0.0)
ia-wiki-Talk	N/A(N/A)	960404(960749.7)	**953079(953135.6)**	7325(7614.1)
inf-power	121386(121503.3)	121684(121806.5)	**120083(120093.7)**	1601(1712.8)
inf-roadNet-CA	N/A(N/A)	58226971(**58242646.1**)	**58200545**(58246127.5)	26426(-3481.4)
inf-roadNet-PA	N/A(N/A)	**31986765(32024783.6)**	32004664(32053911.3)	-17899(-29127.7)
rec-amazon	N/A(N/A)	2621204(2621603.4)	**2614369(2615387.0)**	6835(6216.4)
sc-ldoor	N/A(N/A)	**49451610(49453982.0)**	49468843(49482118.7)	-17233(-28136.7)
sc-msdoor	N/A(N/A)	22024646(22025753.5)	**22014986(22022183.9)**	9660(3569.6)
sc-nasasrb	N/A(N/A)	3005909(3006706.0)	**2999302(3000299.7)**	6607(6406.3)
sc-pkustk11	N/A(N/A)	4895749(4896607.9)	**4883668(4884546.0)**	12081(12061.9)
sc-pkustk13	N/A(N/A)	5194083(5194416.5)	**5187461(5188320.6)**	6622(6095.9)
sc-pwtk	N/A(N/A)	12204037(12204847.3)	**12128684(12136111.9)**	75353(68735.4)
sc-shipsec1	N/A(N/A)	6868276(6871559.7)	**6805463(6808142.1)**	62813(63417.6)
sc-shipsec5	N/A(N/A)	8660292(8663708.8)	**8505853(8511135.0)**	154439(152573.8)
soc-BlogCatalog	N/A(N/A)	1182402(1183086.9)	**1174568(1175283.8)**	7834(7803.1)
soc-brightkite	N/A(N/A)	1185642(1186207.8)	**1174634(1174997.4)**	11008(11210.4)
soc-buzznet	N/A(N/A)	1752048(1753239.3)	**1737874(1739660.0)**	14174(13579.3)
soc-delicious	N/A(N/A)	5001861(5005206.1)	**4913722(4926734.0)**	88139(78472.1)
soc-digg	N/A(N/A)	5996955(5998883.5)	**5953253(5954478.7)**	43702(44404.8)
soc-douban	N/A(N/A)	515633(515665.2)	**515270(515270.0)**	363(395.2)
soc-epinions	N/A(N/A)	539836(539930.3)	**535279(535334.1)**	4557(4596.2)
soc-flickr	N/A(N/A)	8587787(8590799.4)	**8574908(8576170.8)**	12879(14628.6)
soc-flixster	N/A(N/A)	5709915(5711787.2)	**5700669(5700966.7)**	9246(10820.5)
soc-FourSquare	N/A(N/A)	5301489(5303507.9)	**5286968(5288261.5)**	14521(15246.4)
soc-gowalla	N/A(N/A)	4722260(4723711.1)	**4712393(4713046.2)**	9867(10664.9)
soc-lastfm	N/A(N/A)	4663155(4664393.4)	**4647964(4648526.2)**	15191(15867.2)
soc-livejournal	N/A(N/A)	109556028(109565450.5)	**109452822(109477352.5)**	103206(88098.0)
soc-LiveMocha	N/A(N/A)	2500934(2501577.3)	**2486310(2487190.1)**	14624(14387.2)
soc-pokec	N/A(N/A)	**51905133(51948256.9)**	52089306(52174954.5)	-184173(-226697.6)
soc-slashdot	N/A(N/A)	1244680(1245411.3)	**1233605(1233761.6)**	11075(11649.7)
soc-twitter-follows	N/A(N/A)	135820(135825.1)	**135811(135811.0)**	9(14.1)
soc-youtube	N/A(N/A)	8119905(8121101.5)	**8091047(8092993.7)**	28858(28107.8)
soc-youtube-snap	N/A(N/A)	15343369(15361768.5)	**15119700(15132843.9)**	223669(228924.6)
socfb-A-anon	N/A(N/A)	**23721593(23776568.8)**	24532930(24615157.4)	-811337(-838588.6)
socfb-B-anon	N/A(N/A)	**18895603(18961692.8)**	19471922(19536195.0)	-576319(-574502.2)
socfb-Berkeley13	N/A(N/A)	1015206(1015683.8)	**1011564(1012052.0)**	3642(3631.8)
socfb-CMU	296930(297032.5)	293356(293521.0)	**292405(292475.6)**	951(1045.4)
socfb-Duke14	458100(460907.8)	452077(452440.7)	**450608(450884.8)**	1469(1555.9)
socfb-Indiana	N/A(N/A)	1376812(1377516.1)	**1372314(1373628.2)**	4498(3887.9)
socfb-MIT	274242(274443.0)	273364(273433.8)	**272431(272497.1)**	933(936.7)
socfb-OR	N/A(N/A)	2112669(2113789.8)	**2105137(2106559.1)**	7532(7230.7)
socfb-Penn94	N/A(N/A)	1823921(1824315.4)	**1817706(1819186.5)**	6215(5128.9)
socfb-Stanford3	506903(507561.5)	496956(497149.2)	**495141(495278.8)**	1815(1870.4)
socfb-Texas84	N/A(N/A)	1651187(1651791.4)	**1646202(1647091.5)**	4985(4699.9)
socfb-UCLA	913929(915068.0)	892101(892692.4)	**888754(889176.7)**	3347(3515.7)
socfb-UConn	792021(793196.8)	774619(775117.6)	**771323(771968.3)**	3296(3149.3)
socfb-UCSB37	677548(678029.5)	662283(662539.4)	**659310(659816.9)**	2973(2722.5)
socfb-UF	N/A(N/A)	1603061(1603611.0)	**1597610(1598484.3)**	5451(5126.7)
socfb-UIllinois	N/A(N/A)	1416782(1417345.7)	**1412639(1413151.5)**	4143(4194.2)
socfb-Wisconsin87	N/A(N/A)	1074892(1075467.0)	**1071497(1072243.4)**	3395(3223.6)
tech-as-caida2007	N/A(N/A)	201561(201609.6)	**200213(200213.0)**	1348(1396.6)
tech-as-skitter	N/A(N/A)	30595343(30624169.3)	**30189100(30307923.2)**	406243(316246.1)
tech-internet-as	N/A(N/A)	312395(312540.5)	**310196(310196.0)**	2199(2344.5)
tech-p2p-gnutella	N/A(N/A)	919297(919598.5)	**916186(916187.8)**	3111(3410.7)
tech-RL-caida	N/A(N/A)	4227355(4228849.7)	**4195752(4199767.5)**	31603(29082.2)
tech-routers-rf	44919(44936.5)	45159(45170.5)	**44892(44894.3)**	267(276.2)
tech-WHOIS	128568(128588.0)	129035(129062.4)	**128336(128336.9)**	699(725.5)
web-arabic-2005	N/A(N/A)	6570715(6571063.0)	**6556866(6557500.9)**	13849(13562.1)
web-BerkStan	292693(293081.0)	287509(287700.9)	**285504(285547.0)**	2005(2153.9)
web-edu	79499(79545.5)	80068(80104.2)	**79042(79050.3)**	1026(1053.9)
web-google	**27842(27842.0)**	27929(27955.5)	**27842(27842.0)**	87(113.5)
web-indochina-2004	409686(409765.0)	407526(407630.7)	**404490(404533.5)**	3036(3097.2)
web-it-2004	N/A(N/A)	**23795542(23797185.1)**	23809850(23813925.5)	-14308(-16740.4)
web-sk-2005	N/A(N/A)	**3131744(3132244.9)**	3133917(3134504.5)	-2173(-2259.6)
web-spam	129440(129534.8)	129729(129779.4)	**128964(128965.8)**	765(813.6)
web-uk-2005	N/A(N/A)	7562800(7562877.4)	**7562306(7562306.0)**	494(571.4)
web-webbase-2001	144674(144718.5)	144786(144879.4)	**144115(144122.0)**	671(757.4)
web-wikipedia2009	N/A(N/A)	36945789(36959858.9)	**36626752(36663286.9)**	319037(296572.0)

ference of the best or average weighted vertex cover obtained by SARRCCP compared with other algorithms. If an algorithm fails to find a vertex cover within the time limit, the column is marked as "N/A". The bold value indicates the better solution values obtained among the algorithms compared.

Table 3: Running time of MS-ITS, FastWVC, and SARRCCP on the massive graphs.

Instance	MS-ITS	FasrWVC	SARRCCP	Instance	MS-ITS	FasrWVC	SARRCCP
bio-dmela	2126.4	226.7	37.7	soc-lastfm	N/A	869.5	162.3
bio-yeast	53.9	773.3	1.6	soc-livejournal	N/A	1004.9	977.7
ca-AstroPh	16156.9	2.9	543.7	soc-LiveMocha	N/A	935.1	962.2
ca-citeseer	N/A	426.2	236.9	soc-pokec	N/A	1000.0	984.7
ca-coauthors-dblp	N/A	928.5	992.5	soc-slashdot	N/A	994.1	997.0
ca-CondMat	9860.3	4.1	723.7	soc-twitter-follows	N/A	358.1	2.5
ca-CSphd	1.2	<0.01	0.4	soc-youtube	N/A	855.1	822.1
ca-dblp-2010	N/A	409.4	240.2	soc-youtube-snap	N/A	1000.0	990.2
ca-dblp-2012	N/A	638.0	393.3	socfb-A-anon	N/A	1000.0	982.2
ca-Erdos992	8.4	486.6	0.3	socfb-B-anon	N/A	1000.0	983.2
ca-GrQc	674.0	721.1	35.0	socfb-Berkeley13	N/A	460.2	504.8
ca-HepPh	10475.3	1.6	272.1	socfb-CMU	1011.9	615.3	149.8
ca-hollywood-2009	N/A	1000.0	971.6	socfb-Duke14	1629.4	769.4	211.7
ca-MathSciNet	N/A	635.7	306.7	socfb-Indiana	N/A	795.1	690.8
ia-email-EU	N/A	658.9	0.5	socfb-MIT	12093.1	384.9	167.4
ia-email-univ	91.7	274.0	1.3	socfb-OR	N/A	965.3	996.8
ia-enron-large	N/A	7.0	971.0	socfb-Penn94	N/A	871.1	859.8
ia-fb-messages	44.5	436.6	0.7	socfb-Stanford3	4388.6	761.2	294.2
ia-reality	10.4	<0.01	<0.01	socfb-Texas84	N/A	920.7	696.0
ia-wiki-Talk	N/A	26.2	984.1	socfb-UCLA	154.9	892.1	244.5
inf-power	993.3	560.7	37.8	socfb-UConn	15843.7	154.1	317.1
inf-roadNet-CA	N/A	1000.0	998.1	socfb-UCSB37	5456.8	488.7	331.1
inf-roadNet-PA	N/A	1000.0	999.1	socfb-UF	N/A	846.5	619.6
rec-amazon	N/A	946.7	37.2	socfb-UIllinois	N/A	782.0	618.8
sc-ldoor	N/A	999.5	989.7	socfb-Wisconsin87	N/A	513.4	588.1
sc-msdoor	N/A	985.5	782.5	tech-as-caida2007	N/A	79.6	28.6
sc-nasasrb	N/A	990.9	999.2	tech-as-skitter	N/A	1000.0	994.0
sc-pkustk11	N/A	992.1	54.4	tech-internet-as	N/A	2.4	209.2
sc-pkustk13	N/A	981.3	28.1	tech-p2p-gnutella	N/A	332.4	906.3
sc-pwtk	N/A	980.6	669.5	tech-RL-caida	N/A	995.6	993.8
sc-shipsec1	N/A	916.0	336.9	tech-routers-rf	61.0	653.0	5.9
sc-shipsec5	N/A	922.1	692.4	tech-WHOIS	6500.0	365.2	30.5
soc-BlogCatalog	N/A	887.5	706.6	web-arabic-2005	N/A	995.0	584.2
soc-brightkite	N/A	622.0	982.4	web-BerkStan	2304.9	647.4	166.7
soc-buzznet	N/A	975.9	989.8	web-edu	242.3	817.4	48.1
soc-delicious	N/A	888.9	996.0	web-google	0.8	309.2	0.8
soc-digg	N/A	858.5	645.8	web-indochina-2004	3187.9	893.4	319.9
soc-douban	N/A	758.2	13.6	web-it-2004	N/A	990.6	995.7
soc-epinions	N/A	561.9	725.1	web-sk-2005	N/A	948.9	134.6
soc-flickr	N/A	994.6	548.1	web-spam	644.2	915.6	52.7
soc-flixster	N/A	469.9	89.8	web-uk-2005	N/A	962.9	282.0
soc-FourSquare	N/A	786.6	204.6	web-webbase-2001	399.9	710.8	44.7
soc-gowalla	N/A	373.2	169.0	web-wikipedia2009	N/A	1000.0	996.7

7.1 Experiments on Class LPI, BHOSLIB and DIMACS

The results on Class LPI, BHOSLIB and DIMACS are summarized in Table 1. As can be seen from the table, SARRCCP dramatically outperforms its competitor MS-ITS on these benchmarks. The results show that SARRCCP can find better or equal solutions for all instances. Particularly, SARRCCP obtain 20 new best solutions for these instances. For average solution quality, SARRCCP also dominates MS-ITS except two instances (800*5000, C2000.5). Moreover, most instances can be solved by SARRCCP more quickly than MS-ITS, especially for the large-sized graphs.

7.2 Experiments on Massive Graphs

The results on massive graphs are summarized in Table 2. From the table we can observe that, MS-ITS can only provide 27 solutions of the total 86 instances. For these 27 solutions, SARRCCP can obtain better or equal weighted vertex covers than MS-ITS. Additionally, compared with FastWVC, SARRCCP can find better solutions than FastWVC for 75 graphs, while FastWVC can find better solutions for 9 graphs, and for the remaining 2 graphs, both of the algorithms can obtain the same minimum weighted vertex covers. The real time on massive graphs are summarized in Table 3. The running time of SARRCCP is obviously orders of magnitudes less than that of MS-ITS. And for most instances, SARRCCP is faster than FastWVC.

7.3 The effectiveness of RR, CCP and SAVR

To study the effectiveness of reduction rules, CCP strategy and the SAVR heuristic, we compare SARRCCP with three other alternative algorithms named SARRCC, the SACCP and RRCCP. In SARRCC, the CCP strategy is replaced with the original CC strategy; in SACCP, we do not use the reduction rules during the construction phase; and in RRCCP, we use a step-by-step vertex removing strategy rather than self-adaptive vertex removing strategy. The results are summarized in Table 4, which shows that SARRCCP outperforms these three variants on all benchmarks. This indicates that all the strategies proposed in this paper play a key role in our algorithm.

8 Summary and Future Work

In this work, we develop a local search algorithm called SARRCCP for MWVC. Four reduction rules are introduced to obtain a better initial candidate solution. A new variant of configuration checking strategy, as called configuration checking strategy with priority criterion, is proposed to handle the cycling problem in local search. The self-adaptive vertex removing strategy is then proposed to make a balance between intensification search and diversification search. Experimental results show that the SARRCCP algorithm performs much better than state-of-art algorithms. In the future, since the strategies introduced in our algorithm are general, we would like to adapt our algorithm to solve other combinatorial problems. For massive graphs, it is interesting to design other low-complexity heuristics to improve the local search for MWVC.

Table 4: Experiment results of SARRCC,SACCP, RRCCP and SARRCCP on the massive graphs.

Instance	SARRCC Best(Avg)	SACCP Best(Avg)	RRCCP Best(Avg)	SARRCCP Best(Avg)
ca-CondMat	686401(686428.7)	683719(683805.3)	683673(683752.0)	**683659(683745.7)**
ca-CSphd	29407(29407.8)	**29390(29390.0)**	**29390(29390.0)**	**29390(29390.0)**
ca-Erdos992	**28298(28298.0)**	**28298(28298.0)**	**28298(28298.0)**	**28298(28298.0)**
ca-GrQc	122739(122774.7)	122251(122255.8)	122251(122256.0)	**122250(122254.3)**
ca-HepPh	365350(365474.7)	363998(364017.4)	363985(364020.1)	**363976(364001.4)**
ia-email-EU	**48269(48269.0)**	**48269(48269.0)**	**48269(48269.0)**	**48269(48269.0)**
ia-email-univ	32968(33016.6)	**32931(32931.0)**	**32931(32931.0)**	**32931(32931.0)**
ia-fb-messages	32344(32352.2)	**32300(32300.0)**	**32300(32300.0)**	**32300(32300.0)**
ia-reality	**4894(4894.0)**	**4894(4894.0)**	**4894(4894.0)**	**4894(4894.0)**
soc-buzznet	1761408(1761685.4)	1739983(1741523.0)	1738925(1740153.1)	**1737874(1739660.0)**
soc-douban	515322(515358.5)	**515270(515270.0)**	**515270(515270.0)**	**515270(515270.0)**
soc-FourSquare	5296080(5296476.6)	5287049(5289008.0)	5287654(5288544.3)	**5286968(5288261.5)**
soc-pokec	52149467(52248940.3)	52096360(52185116.3)	52122597(52202284.8)	**52089306(52174954.5)**
soc-slashdot	1244240(1244622.9)	1234741(1235493.5)	1233698(1234516.7)	**1233605(1233761.6)**
soc-twitter-follows	**135811**(135817.3)	**135811(135811.0)**	**135811(135811.0)**	**135811(135811.0)**
soc-youtube-snap	15159868(15190478.7)	15459807(15504854.1)	15121193(15156195.5)	**15119700(15132843.9)**
socfb-Duke14	454243(454467.7)	450684(450919.6)	450667(450973.5)	**450608(450884.8)**
socfb-Stanford3	498634(498789.7)	495310(495380.5)	495234(495359.4)	**495141(495278.8)**
tech-as-caida2007	201060(201119.5)	**200213(200213.0)**	**200213(200213.0)**	**200213(200213.0)**
tech-internet-as	311942(312074.1)	**310196**(310196.7)	**310196(310196.0)**	**310196(310196.0)**
tech-RL-caida	4222415(4222986.2)	4200822(4203229.8)	4198649(4200765.5)	**4195752(4199767.5)**
web-google	27887(27912.1)	**27842(27842.0)**	**27842(27842.0)**	**27842(27842.0)**
web-spam	129891(129988.8)	**128964**(128968.0)	**128964**(128966.2)	**128964(128965.8)**
web-uk-2005	**7562306(7562306.0)**	7562624(7562624.8)	**7562306(7562306.0)**	**7562306(7562306.0)**
web-wikipedia2009	36643160(36670314.6)	37018728(37062303.1)	36637574(36685138.9)	**36626752(36663286.9)**

References

[Balachandar and Kannan, 2009] S Raja Balachandar and K Kannan. A meta-heuristic algorithm for vertex covering problem based on gravity. *International Journal of Mathematical and Statistical Sciences*, 1(3):130–136, 2009.

[Balaji *et al.*, 2010] S Balaji, V Swaminathan, and K Kannan. An effective algorithm for minimum weighted vertex cover problem. *International Journal of Computational and Mathematical Sciences*, 4:34–38, 2010.

[Bouamama *et al.*, 2012] Salim Bouamama, Christian Blum, and Abdellah Boukerram. A population-based iterated greedy algorithm for the minimum weight vertex cover problem. *Applied Soft Computing*, 12(6):1632–1639, 2012.

[Cai and Su, 2012] Shaowei Cai and Kaile Su. Configuration checking with aspiration in local search for sat. In *AAAI*, 2012.

[Cai and Su, 2013] Shaowei Cai and Kaile Su. Local search for boolean satisfiability with configuration checking and subscore. *Artificial Intelligence*, 204:75–98, 2013.

[Cai *et al.*, 2010] Shaowei Cai, Kaile Su, and Qingliang Chen. Ewls: A new local search for minimum vertex cover. In *AAAI*, 2010.

[Cai *et al.*, 2011] Shaowei Cai, Kaile Su, and Abdul Sattar. Local search with edge weighting and configuration checking heuristics for minimum vertex cover. *Artificial Intelligence*, 175(9):1672–1696, 2011.

[Cai *et al.*, 2013] Shaowei Cai, Kaile Su, Chuan Luo, and Abdul Sattar. Numvc: An efficient local search algorithm for minimum vertex cover. *Journal of Artificial Intelligence Research*, pages 687–716, 2013.

[Cai *et al.*, 2015] Shaowei Cai, Jinkun Lin, and Kaile Su. Two weighting local search for minimum vertex cover. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[Cai, 2015] Shaowei Cai. Balance between complexity and quality: local search for minimum vertex cover in massive graphs. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI*, pages 25–31, 2015.

[Dinur and Safra, 2005] Irit Dinur and Samuel Safra. On the hardness of approximating minimum vertex cover. *Annals of mathematics*, pages 439–485, 2005.

[Fan *et al.*, 2015] Yi Fan, Chengqian Li, Zongjie Ma, LjiLjana Brankovic, Vladimir Estivill-Castro, and Abdul Sattar. Exploiting reduction rules and data structures: Local search for minimum vertex cover in massive graphs. *arXiv preprint arXiv:1509.05870*, 2015.

[Fang *et al.*, 2014] Zhiwen Fang, Chu-Min Li, Kan Qiao, Xu Feng, and Ke Xu. Solving maximum weight clique using maximum satisfiability reasoning. In *ECAI*, pages 303–308, 2014.

[Johnson and Trick, 1996] David S Johnson and Michael A Trick. *Cliques, coloring, and satisfiability: second DIMACS implementation challenge, October 11-13, 1993*, volume 26. American Mathematical Soc., 1996.

[Jovanovic and Tuba, 2011] Raka Jovanovic and Milan Tuba. An ant colony optimization algorithm with improved pheromone correction strategy for the minimum weight vertex cover problem. *Applied Soft Computing*, 11(8):5360–5366, 2011.

[Karp, 1972] RM Karp. Reducibility among combinatorial problems. complexity of computer computations,(re miller and jm thatcher, eds.), 85–103, 1972.

[Li *et al.*, 2015] Chu-Min Li, Hua Jiang, and Ru-Chu Xu. Incremental maxsat reasoning to reduce branches in a branch-and-bound algorithm for maxclique. In *Learning and Intelligent Optimization*, pages 268–274. Springer, 2015.

[Luo *et al.*, 2014] Cheng Luo, Shanyong Cai, Wenchuan Wu, Zhong Jie, and Kuan-Wu Su. Ccls: An efficient local search algorithm for weighted maximum satisfiability. 2014.

[Luo *et al.*, 2015] Chuan Luo, Shaowei Cai, Kaile Su, and Wei Wu. Clause states based configuration checking in local search for satisfiability. *Cybernetics, IEEE Transactions on*, 45(5):1014–1027, 2015.

[Niedermeier and Rossmanith, 2003] Rolf Niedermeier and Peter Rossmanith. On efficient fixed-parameter algorithms for weighted vertex cover. *Journal of Algorithms*, 47(2):63–77, 2003.

[Richter *et al.*, 2007] Silvia Richter, Malte Helmert, and Charles Gretton. A stochastic local search approach to vertex cover. In *KI 2007: Advances in Artificial Intelligence*, pages 412–426. Springer, 2007.

[Rossi and Ahmed, 2015] Ryan A Rossi and Nesreen K Ahmed. The network data repository with interactive graph analytics and visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[San Segundo *et al.*, 2016] Pablo San Segundo, Alvaro Lopez, and Panos M Pardalos. A new exact maximum clique algorithm for large and massive sparse graphs. *Computers & Operations Research*, 66:81–94, 2016.

[Shyu *et al.*, 2004] Shyong Jian Shyu, Peng-Yeng Yin, and Bertrand MT Lin. An ant colony optimization algorithm for the minimum weight vertex cover problem. *Annals of Operations Research*, 131(1-4):283–304, 2004.

[Voß and Fink, 2012] Stefan Voß and Andreas Fink. A hybridized tabu search approach for the minimum weight vertex cover problem. *Journal of Heuristics*, 18(6):869–876, 2012.

[Wang *et al.*, 2015] Yiyuan Wang, Dantong Ouyang, Liming Zhang, and Minghao Yin. A novel local search for unicost set covering problem using hyperedge configuration checking and weight iversity. *SCIENCE CHINA Information Sciences*, 2015.

[Wang *et al.*, 2016] Yiyuan Wang, Shaowei Cai, and Minghao Yin. Two efficient local search algorithms for maximum weight clique problem. In *AAAI2016*, 2016.

[Xu *et al.*, 2007] Ke Xu, Frédéric Boussemart, Fred Hemery, and Christophe Lecoutre. Random constraint satisfaction: Easy generation of hard (satisfiable) instances. *Artificial Intelligence*, 171(8):514–534, 2007.

[Zhou *et al.*, 2015] Taoqing Zhou, Zhipeng Lü, Yang Wang, Junwen Ding, and Bo Peng. Multi-start iterated tabu search for the minimum weight vertex cover problem. *Journal of Combinatorial Optimization*, pages 1–17, 2015.