Article

An Efficient Heuristic Algorithm for Solving Connected Vertex Cover Problem in Graph Theory

Yongfei Zhang [1], Jun Wu [1], Liming Zhang [2,3], Peng Zhao [1], Junping Zhou [1,2,*] and Minghao Yin [1,2]

[1] College of Information Science and Technology, Northeast Normal University, Changchun, 130117, China
[2] Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Changchun 130012, China
[3] College of Computer Science and Technology, Jilin University Changchun 130012, China
* Correspondence: zhoujp877@nenu.edu.cn

Version January 25, 2018 submitted to

Abstract: The connected vertex cover (CVC) problem is a variant of the vertex cover problem, which has many important applications, such as wireless network design, routing and wavelength assignment problem, etc. A good algorithm for the problem can help us improve engineering efficiency, cost savings and resources in industrial applications. In this work, we present an efficient algorithm GRASP-CVC (Greedy Randomized Adaptive Search Procedure for Connected Vertex Cover) for CVC in general graphs. The algorithm has two main phases, i.e., construction phase and local search phase. To construct a high quality feasible initial solution, we design a greedy function and a restricted candidate list in the construction phase. The configuration checking strategy is adopted to decrease the cycling problem in the local search phase. The experimental results demonstrate that GRASP-CVC is competitive with the other competitive algorithm, which validate the effectivity and efficiency of our GRASP-CVC solver.

Keywords: Heuristic algorithm; connected vertex cover; GRASP.

1. Introduction

The connected vertex cover(CVC) is one of the classical combinatorial optimization problems, which was first introduced by Garey and Johnson in paper [1]. The problem not only shows its great importance in theory, but also has many significant industrial applications [2–5]. For example, in the wireless newtwork design, the vertices of the newtwork are connected by transmission links. We want to place a minimum number of relay stations on vertices such that any pair of relay stations are connected and every transmission link is incident to a relay station. This is the most direct application of the connected vertex cover model in the industry. Designing a good algorithm to solve this problem can not only improve work efficiency, save money and labor cost, but also save natural resources and reduce material waste. The problem is known to be NP-hard even in the planar 2-connected graph of maximum degree 4 [6], planar bipartite graph with maximum degree 4 [7], as well as in 3-connected graph [8].

The CVC problem has been studied for a long time, and a lot of efforts have been devoted to it. To date, there are mainly two types of algorithms to solve CVC, i.e., exact algorithms and approximation algorithms. All existing exact algorithms for CVC are mainly FPT (fixed-parameter tractable) algorithms in theory and these theoretical results are obtained in worst case. For example, Moser [2] showed that CVC is fixed-parameter tractable using the tree width as a parameter and proposed a dynamic programming algorithm running in $O(2^w \cdot w^{3w+2} \cdot n)$ time, where w is the tree width, n is the number of nodes of nice tree decomposition. With the desired vertex cover size k as parameter, Richter et al. [9] proposed an improved algorithm with running time in $O(2.7606^k)$ in the worst case. Binkele-Raible [10] provided a better exact algorithm with running time in $O(2.4882^k)$ in the worst case. Because these exact algorithms fail to solve large graphs, a lot of efforts on approximation algorithms have been devoted. In the general graph, Savage [11] proposed the first constant ratio

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 25 January 2018 doi:10.20944/preprints201801.0243.v1

algorithm and proved that the set of internal nodes of any depth-first search tree is a solution of 2-approximation for CVC problem. In addition, Fujito and Doi [12] proposed a 2-approximation algorithm for solving CVC, which runs in $O(log^2 n)$ time using $O(\delta^2(m+n)/logn)$ processors on an EREW-PRAM, where n is the number of vertices, m the number of edges, and δ is the maximum vertex degree. Fernau and Manlove [7] proved that CVC is NP-hard to approximate within $10\sqrt{5}-21$ in general graphs unless $P = NP$. Therefore, it is difficult to improve the approximation ratio of approximation algorithms in general graphs, which makes the researchers change their research angle into the special graphs. Escoffier et al. [13] proved that the CVC problem is APX-complete in bipartite graphs of maximum degree 4 and is polynomial time solvable in chordal graphs. In addition, they also showed that CVC is 5/3-approximable for a class of special graphs (where solving the minimum vertex cover problem used polynomial time) and a polynomial time approximation algorithm for CVC in planar graphs was presented. Cardinal and Levy [14] proposed an approximation algorithm in dense graphs and the algorithm approximated the CVC problem with a ratio strictly less than 2 in dense graphs. The first polynomial time approximation algorithm in unit disk graphs for CVC problem was proposed in [3]. Li et al.[15] proved that the CVC problem is still NP-hard for 4-regular graphs and provided a lower bound for the problem. Moreover, they proposed two approximation schemes for this problem in 4-regular graphs with approximation ratio $3/2$ and $4/3 + O(1/n)$, respectively. Although the exact algorithms for CVC can provide an optimal solution, they are hard and time consuming to deal with large scale instances. Furthermore, although some approximation algorithms for CVC can get good performance in special graphs, they are usually not suitable for dealing with general graphs, and the state-of-the-art approximation methods in general graphs can only provide an approximate ratio 2, which is often not enough in practice. This yields a new challenge for us to devise a heuristic algorithm for CVC that can deal with large general graphs and obtain the best possible approximate solutions within a reasonable time.

In this article, the heuristic algorithm GRASP-CVC for CVC in general graphs is proposed and this algorithm can obtain a relatively good solution within a reasonable time. The heuristic algorithm GRASP-CVC is based on the framework of greedy randomized adaptive search procedure ($GRASP$) [16]. The algorithm GRASP-CVC has two main phases, i.e., construction phase and local search phase. The GRASP-CVC tries to construct a feasible initial solution greedily in the construction phase. During this phase, we design a greedy function to help to evaluate the benefit of adding a vertex to the current solution. Besides, we construct a restricted candidate list (RCL) to assist in constructing a high quality initial solution in the construction phase. Then the initial solution is further improved in the local search phase. To prevent the local search from suffering severe cycling problem, the configuration checking (CC) strategy is adopted in the search. Relying on the CC, we avoid many unnecessary searches during the local search procedure and the efficiency of the GRASP-CVC is improved greatly. Once the local search phase cannot explore a better solution anymore, which means the local search phase reaches a local optima, the GRASP-CVC then restarts a new iteration and repeats the construction and local search phases until reaching the maximum iteration times. The best found solution will be the final solution after all iterations used up. The results of experiments demonstrate that GRASP-CVC provides better solutions compared to the competitive algorithm, which validate the effectivity and efficiency of our GRASP-CVC solver. Moreover, the GRASP-CVC obtains almost the same size solutions in 10 times running, which demonstrates GRASP-CVC is stable.

The rest of this paper is structured as follows. Some relevant definitions and background knowledge will be introduced in next section. In Section 3, the algorithm GRASP-CVC will be introduced and the two main components will be discussed in details. Experimental evaluations and analyses will be shown in Section 4. Conclusions and future work will be given in the last section.

2. Preliminaries

In this section, some definitions and background knowledge are provided. From now on, unless otherwise stated, we only consider the CVC problem on an undirected graph $G = (V, E)$, where

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 25 January 2018 doi:10.20944/preprints201801.0243.v1

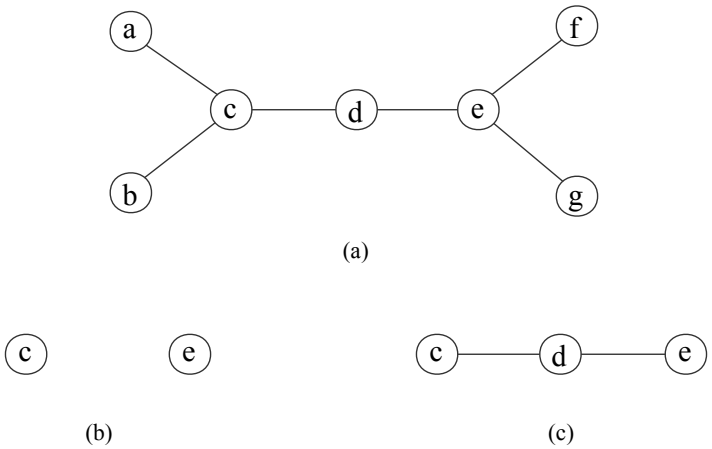

Figure 1. An example for MVC and CVC.

$V = \{v_1, v_2, ..., v_n\}$ is the vertices set and $E = \{e_1, e_2, ..., e_m\}$ is the edges set. In addition, each edge $e_i = (v_k, v_j)(1 \leqslant i \leqslant m, 1 \leqslant k, j \leqslant n, k \neq j)$ is a 2-element tuple on V and we define vertices v_k and v_j are the endpoints of edge e_i. For a vertex subset $C \subseteq V$ and an edge e_i , if C contains no endpoint of e_i, we say e_i is uncovered by C, otherwise, we say e_i is covered by C.

Definition 1. *(Vertex Cover, VC) Given a graph $G = (V, E)$, a subset of vertices $C \subseteq V$ is a vertex cover(VC) of G if each edge in E has at least one endpoint in C.*

Definition 2. *(Minimum Vertex Cover, MVC) Given a graph $G = (V, E)$, the minimum vertex cover (MVC) problem is to compute a VC of minimum cardinality in G.*

Definition 3. *(Induced Sub-graph, IS) Given two graph $G = (V, E)$ and $G' = (V', E')$ where $V' \subseteq V$ and $E' = \{(v, u) \big| v, u \in V' \wedge (v, u) \in E\}$, G' is called an induced sub-graph (IS) of G.*

Definition 4. *(Connected Graph) A graph is a connected graph if there is a path between every pair of vertices.*

Definition 5. *(Connected Vertex Cover, CVC) Given a connected graph $G = (V, E)$, the connected vertex cover (CVC) problem is to determine a subset $C \subseteq V$ with minimum cardinality such that C satisfies the following two conditions: 1) C is a vertex cover; 2) the IS induced by C is a connected graph.*

In order to facilitate the readers to understand the concepts given above, we provide an example in Figure 1. Figure 1(a) is a graph $G = (V, E)$, where $V = \{a, b, c, d, e, f, g\}$ and $E = \{(a, c), (b, c), (c, d), (d, e), (e, f), (e, g)\}$. Figure 1(b) presents an induced sub-graph by the vertex set $V_1' = \{c, e\}$, which is just the solution of the MVC problem of G. Because the induced sub-graph in Figure 1(b) is not connected, the vertex set $V_1' = \{c, e\}$ is not the solution of the CVC problem of G. By adding a vertex d to V_1', we obtain a new vertex set $V_2' = \{c, d, e\}$. The sub-graph induced by V_2' is shown in Figure 1(c). From Figure 1(c), we can notice that the vertex set V_2' is the minimal vertex subset that satisfies: 1) sub-graph induced by V_2' is connected; 2) the vertex set V_2' covers all edges in E. Thus, V_2' is a solution for the CVC problem of G. From the example, we see that the size of the optimal MVC solution provides a lower bound for the size of the optimal CVC solution. In the following, we will present the conclusion in Theorem 1.

Theorem 1. *The size of the optimal MVC solution provides a lower bound for the size of the optimal CVC solution.*

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 25 January 2018 doi:10.20944/preprints201801.0243.v1

Proof. Given an undirected graph G, we suppose the size of the optimal MVC solution is N and the size of the optimal CVC solution is M. Then we will analyze the theorem cases individually.
Case 1. There is a connected optimal solution of MVC. Under this circumstance, the MVC solution is also a CVC solution. Thus, we have $M = N$.
Case 2. There is no connected solution among all of the optimal solutions of MVC. Under this condition, it is impossible that $M = N$. Then we will prove the conclusion by using reduction to absurdity. Suppose there exists a CVC solution that $M \leqslant N$. Under this condition, according to the definitions of CVC and MVC, we know the CVC solution is an MVC solution as well. So, we get one MVC solution whose size is less than N. However, this is contradiction with the previous assumption that N is the size of the optimal MVC solution.
In total, we finally reach the conclusion that $M \geqslant N$, which means that the optimal MVC solution size provides a lower bound for the size of the optimal CVC solution. □

3. GRASP-CVC algorithm for CVC

GRASP-CVC (Greedy Randomized Adaptive Search Procedures for Connected Vertex Cover problem) is a multi-start meta-heuristic, which consists of two main phases: construction phase and local search phase. An initial solution is constructed firstly in the construction phase, and then the local search phase attempts to find the existence of a better solution by exploring the neighborhood of the initial solution. The two phases are executed repeatedly until reaching the termination condition, and then the GRASP-CVC takes the best found solution as the final output. The pseudo code of GRASP-CVC is outlined in Algorithm 1. At first, the solution C^* is initialized (line 1). Then the algorithm enters an iteration loop (line 2-8). In each iteration, an initial CVC solution C is generated firstly by the construction procedure (line 3), and then in the local search procedure (line 4), GRASP-CVC starts its search from C trying to find a better solution. If C possesses fewer vertices than the current best solution C^*, C^* will be updated by C (line 5-6). When the GRASP-CVC reaches the maximum iteration times, C^* will be returned as the final solution (line 9). During the construction, we design a greedy function and construct a restricted candidate list to help to construct a high quality feasible solution. Moreover, in the local search, we adopt the configuration checking to reduce the cycling problem. The two phases will be discussed in details in the next two subsections.

Algorithm 1: GRASP-CVC($MaxIter, seed$)

```
1 initialize the solution C*;
2 for i = 1 to MaxIter do
3 |   C = GreedyConstruction(seed);
4 |   C = LocalSearch(C);
5 |   if |C| < |C*| then
6 |   |   C* = C;
7 |   end
8 end
9 return C*
```

3.1. Construction phase

Before introducing the construction phase, we shall give the definitions of greedy function and restricted candidate list (RCL) that play important roles in the construction phase.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 25 January 2018 doi:10.20944/preprints201801.0243.v1

3.1.1. Greedy function and RCL

To evaluate the benefit of adding v to the current solution, a greedy function $score(v)$ is designed, which is quite important for the construction of RCL as well. In order to introduce the greedy function, we firstly propose some relevant definitions. We say vertices u and v are neighbors each other if there is an edge between them, and we use $N(u) = \{v|(u,v) \in E\}$ denoting the neighbors set of vertex u. The neighbor vertices set of a solution C, denoted as $N(C)$, can be calculated as follows.

$$N(C) = \{v|v \notin C, v \in N(u), u \in C, N(u) = \{v|(u,v) \in E\}\} \tag{1}$$

For a given vertex v, the greedy function $score(v)$ can be calculated by the Formula (2).

$$score(v) = cost(C) - cost(C') \tag{2}$$

In the Formula (2), C is the current solution. If v in C, $C' = C \backslash \{v\}$ ('$\backslash$' means removing the vertex v from C), otherwise, $C' = C \cup \{v\}$. Moreover, $cost(C)$ is also a function calculated by the Formula (3).

$$cost(C) = |\{e|e \text{ is not covered by } C\}| \tag{3}$$

From the formula, we can know that the function $cost(C)$ is to compute the total number of edges uncovered by C. In addition, when v in C, $score(v)$ is a negative number.

Using the greedy function $score(v)$, we can identify those vertices that are most beneficial to the current solution. Moreover, the greedy function is an indispensable part in the construction of RCL.

The RCL consists of the vertices that are most beneficial to the current solution C. In the construction phase, one vertex is chosen randomly from RCL. And to construct a feasible initial solution, we select vertices from RCL. The construction of RCL is described in Formula (4)

$$RCL = \{v|v \in N(C) \wedge score(v) \geqslant score(u), \forall\, u \in N(C)\} \tag{4}$$

Clearly, the elements of RCL are the vertices set that having the highest $score$ in the premise of not destroying the connectivity of C.

3.1.2. Construction procedure

After the necessary descriptions of greedy function and RCL, we shall discuss the greedy construction procedure in details. The greedy construction procedure GreedyConstruction is outlined in Algorithm 2.

Algorithm 2: GreedyConstruction(*seed*)

```
1  C = Φ;
2  RCL = {v|v ∈ V ∧ score(v) ≥ score(u), ∀u ∈ V};
3  initialize the score of each vertex according to Formula(2);
4  while C is not a connected vertex cover do
5  |  choose a vertex v from RCL randomly;
6  |  C = {v} ∪ C;
7  |  update score of each vertex;
8  |  RCL = {v|v ∈ N(C) ∧ score(v) ⩾ score(u), ∀u ∈ N(C)};
9  end
10 return C
```

In the beginning, the connected vertex cover C is initialized (line 1) and the $score$ of each vertex is initialized according to the Formula (2) (line 2). The RCL is initialized to the vertices with the highest $score$ among the vertices set V (line 3). Then, the procedure enters the main loop (line 4-8). In each

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 25 January 2018 doi:10.20944/preprints201801.0243.v1

loop, a vertex v is chosen from the RCL randomly and added into the construction solution C (line 5-6). After the operations of line 5 and 6, the *score* of each vertex is updated according to the Formula (2) (line 7). At the end of the loop, the RCL is updated (line 8). The loop is executed until C is constructed to be a solution of CVC, and then C will be returned at the end of the construction procedure (line 10).

3.2. Local search phase

In this subsection, we shall introduce the configuration checking (CC) strategy and discuss the working process of the local search phase in details.

3.2.1. Configuration checking strategy

Greedy strategy is usually an important part in the local search algorithms. It helps to lift the performance of the local search algorithms on large and hard instances. However, the greedy strategy usually also makes the local search algorithms easier to fall into the cycling problem (which means the algorithm visits the same part of the solution space repeatedly). Up to now, many efficient strategies have been used to handle this problem [17–20]. Configuration checking (CC) [21] strategy is one of those strategies and has been applied to some problems successfully [21–25]. Therefore, we adopt the CC strategy to avoid the cycling problem in the local search.

Before introducing the CC, we shall give the concept of vertex state. The vertex state of a vertex v refers to the fact whether v is located in the current solution. We can use a Boolean value 1 to represent v is in the current solution and 0 to represent is not in. The configuration of a vertex v is the states of all its neighbor vertices, which can be denoted by a n-dimensional Boolean vector c_v(where n is the number of neighbor vertices of v).

The main idea of the CC strategy is that a vertex v is forbidden to add back to the current solution if its configuration keeps unchanged since it was removed from the current solution last time. This strategy is intuitive and reasonable in avoiding cycling problem, as it prevents the search facing the same scenario again. For example, for a $v \in C$, suppose its configuration is $c_{v0} = (0, 1, 0, 1)$ after removing v out of C, then after several steps of searching, v is selected again according to the greedy function value and suppose its configuration is c_{v1} now. If $c_{v0} = c_{v1}$,i.e. configuration not changed, if add v back to the C, then the search goes back to the same situation before last time removing v out of C and the same solution space will be searched repeatedly. However, this situation will not occur when $c_{v0} \neq c_{v1}$, i.e. configuration changed(e.g. $c_{v1} = (1, 1, 0, 1)$). A Boolean array *Change* is used to implement the CC strategy. The element in the array *Change* is *Change*[v], which denotes whether the configuration of vertex v is changed since it was removed from the current solution last time. We use *Change*[v] = 1 expressing change, and *Change*[v] = 0 expressing no change. Only the vertex v whose *Change*[v] = 1 is allowed to be added to the current solution in the local search process. In the process of local search, the values of *Change* are updated according the rules below [21].

- Rule 1: In the beginning, for each vertex v, set *Change*[v] to 1.
- Rule 2: When removing v from C, reset *Change*[v] to 0.
- Rule 3: When u changes its state, for each $v \in \mathrm{N}(u) \backslash \mathrm{C}$, *Change*[$v$] is set to 1.

3.2.2. Local search procedure

In this subsection, the local search procedure is discussed at length. The main steps of the LocalSearch are listed in Algorithm 3.

The local search procedure performs as follows. In the first place, all elements of the *Change* are initialized to 1 (line 1), which means that all vertices are allowed to be added to the current solution in the beginning. Next, the local optimal solution C^* is initialized as C, where C is generated by the construction phase (line 2). Then, in the loop (line 3-17), the LocalSearch(C) checks the feasibility of the current solution C. If C is a CVC solution which has a smaller vertex number than C^*, then C^* will be updated by C and a vertex v with the highest *score* will be removed from C, and then the procedure

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 25 January 2018 doi:10.20944/preprints201801.0243.v1

Algorithm 3: LocalSearch(C)

```
1  initialize each element value of Change array to 1;
2  C* = C;
3  while true do
4      if C is a vertex cover then
5          if C is connected then
6              if |C| < |C*| then
7                  C* = C;
8              end
9          else
10             return C*;
11         end
12         drop a vertex u with the highest score from C and update the Change array;
13         continue;
14     end
15     choose an uncovered edge e randomly;
16     choose a vertex v ∈ e such that Change[v] = 1 with a higher score, C = {v} ∪ C and
         update the Change array;
17 end
```

starts the next cycle (line 4-14). If C is a VC but not a CVC anymore after v is removed, which means the procedure reaches a local optimal solution, then C^* will be returned as the final result of the local search procedure (line 9-11). If C is not a VC anymore after v is removed, the procedure chooses an uncovered edge e randomly, then chooses a vertex $v \in e$ such that $Change[v]$=1 with a higher *score*, and adds to $C = \{v\} \cup C$ (line 15-16). The *Change* array is updated according to Rule 2 and Rule 3 when a vertex changes its state (line 12,16).

3.3. Time complexity analysis

In this subsection, we discuss the time complexity of the main components of the GRASP-CVC algorithm.

First, we consider the algorithm for constructing the initial solution. In each loop, we need to update the *score* for $|V|$ vertices, and scan $|N(C)|$ vertices in order to update the RCL. If every time we add a vertex to C, an average of d edges are covered. Then, we can get a solution in $O(|E|/d * (|V| + |N(C)|)) = O(|E| * |V|)$.

Next, we consider the time complexity of the local search algorithm. In this part, there are three operations that affect the running time of the algorithm: *drop a vertex from C* (Algorithm 3, line 12), *choose an uncovered edge* (Algorithm 3, line 15), and *update the Change array* (Algorithm 3, line 12 and 16). Since the number of vertex in C is $|C|$, the first operation can be done in a time of $O(|C|)$. In our implementation, we maintain a set to record uncovered edges, so an uncovered edge can be chosen in $O(1)$. The time complexity of the third operation depends on the degree of the operated vertex, and therefore this work can be done in O(δ), where δ is the maximum degree of the vertices. Thus, the time complexity of the local search is $O(|C| + 1 + \delta) = O(|C| + \delta)$.

Overall, the run-time complexity of the GRASP-CVC is $O(|E| * |V| + |C| + \delta)$.

4. Computational experiments

During this section, the effectivity and efficiency of the GRASP-CVC algorithm are evaluated by performing some comparison experiments. We compare the GRASP-CVC algorithm with the currently best approximate algorithm (2-approximation algorithm) we know in general graphs [12]. Owing to the researches on CVC problem mainly focused on theoretical studies, there is no available

 doi:10.20944/preprints201801.0243.v1

approximation CVC solver, so we implement the 2-approximation algorithm proposed in [12]. We implement the algorithms GRASP-CVC, and the 2-approximation algorithm in the C++ programming language. All of the experiments are carried on a work station under windows 7 operating system, 3.30GHZ CPU and 8GB memory.

4.1. Benchmark instances

In the experiments, we choose two well-known benchmarks in the field of MVC research, the DIMACS and BHOSLIB instance sets. DIMACS benchmark contains both structured and randomized instances. The structured instances are generated from practical problems, such as coding theory, Keller conjecture and so on. The randomized instances are generated from the stochastic models, such as brock instances. The scale of these problem instances is from 50 vertices and 1000 edges to more than 5500 vertices and 5 million edges. BHOSLIB benchmark is famous for its hardness. The benchmark instances are transformed from SAT instances that are generated in the phase transition area. And the instances in the phase transition area have been proved hard. From the two benchmarks, we select 37 DIMACS benchmark instances and 40 BHOSLIB benchmark instances, which are all employed in the best MVC solver [25].

4.2. Experiment parameter settings

Before reporting the experimental results, we shall introduce some parameter settings.

In GRASP-CVC, there are two main parameters: maximum number of iterations ($MaxIter$) and random seed ($seed$). According to our experimental experience, we set $MaxIter$ to 5000 and random seed to an interval from 1 to 10. In order to evaluate the robustness of GRASP-CVC, we execute ten times for each instance using different random seeds.

For the sake of comparison between GRASP-CVC and 2-approximation algorithm for CVC, we execute ten times independent runs for the approximation algorithm as well. In addition, since the 2-approximation algorithm runs very fast, there is no need to set the cut-off time for the 2-approximation algorithm, we just set the cut-off time for GRASP-CVC to 1000 seconds.

4.3. Experimental results

In this subsection, we will provide the computational results of GRASP-CVC (GRASP-CVC) and the 2-approximation algorithm (2-Aprox) on the two chosen benchmarks. In the results, we provide the following information: the number of vertices and edges of each instance ($|V|, |E|$), the best known size of the MVC solution (MVC), the best solution size solved by the corresponding algorithms ($best$), the average size of solutions solved by the corresponding algorithms (avg), the average time consumed by the corresponding algorithms ($time$). In addition, the MVC size with star (*) has been proved to be the optimal size of MVC solution.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 25 January 2018 doi:10.20944/preprints201801.0243.v1

Table 1. Experimental result on DIMACS Instances

Instance	$\|V\|,\|E\|$	MVC	2-Aprox			GRASP-CVC		
			time	best	avg	time	best	avg
*brock*200_2.*mis*	200,10024	188*	0.0003	198	199.7	0.0388	**190**	**190**
*brock*200_4.*mis*	200,6811	183*	0.0003	195	197.2	1.3976	**184**	**184**
*brock*400_2.*mis*	400,20014	371*	0.0007	396	397.5	6.1956	**376**	**376**
*brock*400_4.*mis*	400,20035	367*	0.0006	396	396.6	10.2863	**376**	**376**
*brock*800_2.*mis*	800,111434	776*	0.0017	798	798.2	35.0407	**780**	**780**
*brock*800_4.*mis*	800,111957	774*	0.0017	798	798.1	173.9981	**780**	**780**
*C*125.9.*mis*	125,787	91*	0.0002	116	117.4	0.003	**91**	**91**
*C*250.9.*mis*	250,3141	206*	0.0005	240	243.2	2.3818	**207**	**207**
*C*500.9.*mis*	500,12418	443*	0.0006	491	494.2	12.051	**448**	**448**
*C*1000.9.*mis*	1000,49421	932	0.0012	994	995.7	66.33	**939**	**939**
*C*2000.9.*mis*	2000,199468	1920	0.004	1992	1992.3	96.854	**1933**	**1933**
*C*2000.5.*mis*	2000,999164	1984	0.0085	1998	1998	942.5859	**1986**	**1986.1**
*C*4000.5.*mis*	4000,3997732	3982	0.0313	3998	3998.2	400.23	**3986**	**3986**
*DSJC*500.5.*mis*	500,62126	487*	0.001	498	499.6	2.419	**487**	**487**
*DSJC*1000.5.*mis*	1000,249674	985*	0.0025	998	999.8	887.5232	**986**	**986**
*gen*200_*p*0.9_44.*mis*	200,1990	156*	0.0004	189	192.8	0.0583	**164**	**164**
*gen*200_*p*0.9_55.*mis*	200,1990	145*	0.0003	186	191.5	0.0607	**156**	**156**
*gen*400_*p*0.9_55.*mis*	400,7980	345*	0.0007	388	391.5	72.4805	**358**	**358**
*gen*400_*p*0.9_65.*mis*	400,7980	335*	0.0006	389	390.9	18.8135	**354**	**354**
*gen*400_*p*0.9_75.*mis*	400,7980	325*	0.0009	391	393.1	2.8003	**357**	**357**
*hamming*8 − 4.*mis*	256,11776	240*	0.0005	255	255.9	0.0154	**240**	**240**
*hamming*10 − 4.*mis*	1024,89600	984*	0.0017	1023	1023.5	273.0134	**990**	**990**
*keller*4.*mis*	171,5100	160*	0.0003	170	170.1	3.9266	**160**	**160**
*keller*5.*mis*	776,74710	749*	0.0015	775	775.6	4.6351	**756**	**756**
*keller*6.*mis*	3361,1026582	3302	0.0107	3360	3360.1	82.1559	**3324**	**3324**
*MANN_a*27.*mis*	378,702	252*	0.0007	290	293.8	0.0475	**260**	**260**
*MANN_a*45.*mis*	1035,1980	690*	0.0011	765	766.4	0.8462	**704**	**704**
*MANN_a*81.*mis*	3321,6480	2221	0.0034	2353	2357.6	5.1007	**2241**	**2241**
*p_hat*300 − 1.*mis*	300,33917	292*	0.0006	298	298	0.1078	**292**	**292**
*p_hat*300 − 2.*mis*	300,22922	275*	0.0004	298	298	0.642	**275**	**275**
*p_hat*300 − 3.*mis*	300,11460	264*	0.0003	294	296	0.338	**264**	**264**
*p_hat*700 − 1.*mis*	700,183651	689*	0.0019	698	699.8	7.7827	**689**	**689**
*p_hat*700 − 2.*mis*	700,122922	656*	0.0017	700	700	1.792	**656**	**656**
*p_hat*700 − 3.*mis*	700,61640	638*	0.0013	694	695.9	176.3111	**638**	**638**
*p_hat*1500 − 1.*mis*	1500,839327	1488*	0.0074	1500	1500	19.7254	**1489**	**1489.1**
*p_hat*1500 − 2.*mis*	1500,555290	1435*	0.0051	1498	1498.1	27.6126	**1438**	**1438**
*p_hat*1500 − 3.*mis*	1500,277006	1406	0.0033	1496	1496	210.4793	**1409**	**1409**

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 25 January 2018 doi:10.20944/preprints201801.0243.v1

Table 2. Experimental result on BHOSLIB Instances

Instance	$\|V\|,\|E\|$	MVC	2-Aprox			GRASP-CVC		
			time	best	avg	time	best	avg
$frb30-15-1.mis$	450,17827	420*	0.001	449	449.6	11.3735	**424**	**424**
$frb30-15-2.mis$	450,17874	420*	0.0006	447	448.4	7.873	**425**	**425**
$frb30-15-3.mis$	450,17809	420*	0.0007	449	449.7	13.4266	**424**	**424**
$frb30-15-4.mis$	450,17831	420*	0.0006	448	449.3	15.2355	**424**	**424**
$frb30-15-5.mis$	450,17794	420*	0.0005	448	449.3	25.6118	**423**	**423**
$frb35-17-1.mis$	595,27856	560*	0.0011	592	593.7	29.4486	**565**	**565**
$frb35-17-2.mis$	595,27847	560*	0.0009	592	592.8	8.5406	**565**	**565**
$frb35-17-3.mis$	595,27931	560*	0.001	592	592.7	2.7005	**565**	**565**
$frb35-17-4.mis$	595,27842	560*	0.001	592	593.6	172.9833	**565**	**565**
$frb35-17-5.mis$	595,28143	560*	0.0009	594	594.5	34.4835	**565**	**565**
$frb40-19-1.mis$	760,41314	720*	0.0012	758	759	187.7404	**728**	**728**
$frb40-19-2.mis$	760,41263	720*	0.0012	757	758	213.6616	**726**	**726**
$frb40-19-3.mis$	760,41095	720*	0.0012	758	759.2	101.0795	**725**	**725**
$frb40-19-4.mis$	760,41605	720*	0.0012	757	758.1	8.3877	**726**	**726**
$frb40-19-5.mis$	760,41619	720*	0.0014	758	759	33.3235	**725**	**725**
$frb45-21-1.mis$	945,59186	900*	0.0015	944	944.4	665.5343	**908**	**908**
$frb45-21-2.mis$	945,58624	900*	0.0017	942	943.3	83.904	**908**	**908**
$frb45-21-3.mis$	945,58245	900*	0.0015	941	943	479.1447	**908**	**908**
$frb45-21-4.mis$	945,58549	900*	0.0019	941	942.6	135.2191	**907**	**907**
$frb45-21-5.mis$	945,58579	900*	0.0019	943	943.7	235.1495	**909**	**909**
$frb50-23-1.mis$	1150,80072	1100*	0.0016	1146	1147.6	409.0972	**1109**	**1109**
$frb50-23-2.mis$	1150,80851	1100*	0.0018	1147	1148.6	87.4731	**1110**	**1110**
$frb50-23-3.mis$	1150,81068	1100*	0.0021	1147	1148.1	414.1894	**1110**	**1110**
$frb50-23-4.mis$	1150,80258	1100*	0.002	1147	1148.5	431.3235	**1110**	**1110**
$frb50-23-5.mis$	1150,80035	1100*	0.002	1147	1148.3	837.1014	**1109**	**1109**
$frb53-24-1.mis$	1272,94227	1219*	0.002	1269	1270.7	695.5386	**1230**	**1230**
$frb53-24-2.mis$	1272,94289	1219*	0.0022	1269	1270.4	390.4401	**1230**	**1230**
$frb53-24-3.mis$	1272,94127	1219*	0.002	1270	1271.2	552.5761	**1229**	**1229**
$frb53-24-4.mis$	1272,94308	1219*	0.0024	1270	1270.4	588.6573	**1229**	**1229**
$frb53-24-5.mis$	1272,94226	1219*	0.0022	1270	1270.6	95.6667	**1230**	**1230**
$frb56-25-1.mis$	1400,109676	1344*	0.0022	1398	1399.1	90.5451	**1357**	**1357**
$frb56-25-2.mis$	1400,109401	1344*	0.0022	1397	1397.3	412.38	**1353**	**1353**
$frb56-25-3.mis$	1400,109379	1344*	0.0023	1397	1398.6	190.6417	**1356**	**1356**
$frb56-25-4.mis$	1400,110038	1344*	0.0025	1398	1399	47.1687	**1356**	**1356**
$frb56-25-5.mis$	1400,109601	1344*	0.0023	1397	1398.4	616.9286	**1355**	**1355**
$frb59-26-1.mis$	1534,126555	1475*	0.0027	1533	1533.2	656.1494	**1487**	**1487**
$frb59-26-2.mis$	1534,126163	1475*	0.0024	1531	1532.5	100.0605	**1488**	**1488**
$frb59-26-3.mis$	1534,126082	1475*	0.0026	1531	1532.5	495.5281	**1489**	**1489**
$frb59-26-4.mis$	1534,127011	1475*	0.0027	1531	1532.4	319.4073	**1487**	**1487**
$frb59-26-5.mis$	1534,125982	1475*	0.0026	1533	1533.3	662.7202	**1487**	**1487**

In Table 1, we present the experimental results on DIMACS benchmark. As is shown in the table, compared to the 2-Aprox algorithm, GRASP-CVC finds the better quality CVC solutions on all the 37 DIMACS instances. On most instances, the solutions found by GRASP-CVC are very close to the optimal MVC and it even finds the same size solutions as optimal MVC on 10 instances in a very short time, which means it finds the optimal CVC solutions on these 10 instances. Besides, the GRASP-CVC consumes very little time on most of the instances, which indicates the efficiency of the GRASP-CVC. Moreover, the GRASP-CVC takes a relatively long time on several larger instances (such as $C2000.5.mis$,DSJC1000.5.mis, $C4000.5.mis$ and so on), this shows that the algorithm still has room for improvement. Another point worth noting is that though the 2-Aprox takes the less time on all of the 37 instances, most of its solutions are not so satisfactory compared to GRASP-CVC.

Table 2 provides the results on BHOSLIB benchmark. We can get the same conclusion as DIMACS that the GRASP-CVC performs best, even though it takes much more time than 2-Aprox. The solutions found by GRASP-CVC are close to the optimal MVC, which means the GRASP-CVC is very effective. The columns *best* and *avg* of GRASP-CVC have the same values on almost all of the instances, which means that the GRASP-CVC gets almost the same size solutions in ten times runs and this demonstrates the stability of the GRASP-CVC.

The comparison and experimental analyses above show that the GRASP-CVC has very good effectivity and efficiency for CVC problem. It performs better than the competitive algorithm in solution quality. Moreover, the GRASP-CVC gets almost the same size solutions in ten times runs, which demonstrates the stability of the GRASP-CVC.

5. Conclusion

In this paper, a heuristic algorithm GRASP-CVC for connected vertex cover problem was proposed. A greedy function and a restricted candidate list (RCL) were proposed to help to construct a high quality initial solution. Furthermore, the configuration checking (CC) strategy was employed to reduce the cycling problem and improve the efficiency of the search. Experimental results demonstrate that GRASP-CVC works better than the comparison algorithm, which validates the effectiveness and efficiency of our GRASP-CVC solver. In the future, we will further study various heuristic methods and hope to design a more powerful heuristic algorithm to deal with CVC.

Acknowledgments: This work was fully supported by the National Natural Science Foundation of China under Grant No.61370156, No. 61403076, and No. 61403077; Research Fund for the Doctoral Program of Higher Education No. 20120043120017; Program for New Century Excellent Talents in University No. NCET-13-0724; the Large-scale Scientific Instrument and Equipment Sharing Project of Jilin Province (20150623024TC-03);The Natural Science Foundation for Youths of JiLin Province (20160520104JH).

References

1. Garey, M. R., Johnson, D. S. The rectilinear Steiner tree problem is NP-complete. SIAM Journal on Applied Mathematics,1997, 32(4), 826-834.
2. Moser, H. Exact algorithms for generalizations of vertex cover. Master's thesis, Fakultät für Mathematik und Informatik, Friedrich-Schiller-Universität Jena,2005.
3. Zhang, Z., Gao, X., Wu, W. PTAS for connected vertex cover in unit disk graphs. Theoretical Computer Science, 2009, 410(52), 5398-5402.
4. Guo, P., Wang, J., Geng, X. H., Kim, C. S., Kim, J. U. A variable threshold-value authentication architecture for wireless mesh networks. Journal of Internet Technology, 2014, 15(6), 929-935.
5. Shen, J., Tan, H. W., Wang, J., Wang, J. W., Lee, S. Y. A novel routing protocol providing good transmission reliability in underwater sensor networks. Journal of Internet Technology, 2015, 16(1), 171-178.
6. Priyadarsini, P. L. K., Hemalatha, T. Connected vertex cover in 2-connected planar graph with maximum degree 4 is NP-complete. International Journal of Mathematical, Physical and Engineering Sciences, 2008, 2(1), 51-54.
7. Fernau, H., Manlove, D. F. Vertex and edge covers with clustering properties: Complexity and algorithms. Journal of Discrete Algorithms, 2009, 7(2), 149-167.

8. Watanabe, T., Kajita, S., Onaga, K. Vertex covers and connected vertex covers in 3-connected graphs. IEEE International Sympoisum on Circuits and Systems. IEEE, 1991, vol.2, 1017-1020.
9. Mölle, D., Richter, S., Rossmanith, P. Enumerate and expand: Improved algorithms for connected vertex cover and tree cover. Theory of Computing Systems, 2008, 43(2), 234-253.
10. Binkele-Raible, D. Amortized analysis of exponential time-and parameterized algorithms: Measure & Conquer and Reference Search Trees. Praca doktorska, University of Trier, Trier, Germany,2010.
11. Savage, C. Depth-first search and the vertex cover problem. Information Processing Letters, 1982, 14(5), 233-235.
12. Fujito, T., Doi, T. A 2-approximation NC algorithm for connected vertex cover and tree cover. Information processing letters, 2004, 90(2), 59-63.
13. Escoffier, B., Gourvès, L., Monnot, J. Complexity and approximation results for the connected vertex cover problem in graphs and hypergraphs. Journal of Discrete Algorithms, 2010, 8(1), 36-49.
14. Cardinal, J., Levy, E. Connected vertex covers in dense graphs. Lecture Notes in Computer Science, 2008, 5171, 35-48.
15. Li, Y., Yang, Z., Wang, W. Complexity and algorithms for the connected vertex cover problem in 4-regular graphs. Applied Mathematics and Computation,2017, 301, 107-114.
16. Resende, M. G., Ribeiro, C. C. GRASP: Greedy randomized adaptive search procedures. In Search methodologies, 2014, pp. 287-312, Springer US.
17. Zhou, Y., Zhang, H., Li, R., Wang, J. Two local search algorithms for partition vertex cover problem. Journal of Computational and Theoretical Nanoscience, 2016, 13(1), 743-751.
18. Li, X., Zhang, J., Yin, M. Animal migration optimization: an optimization algorithm inspired by animal migration behavior. Neural Computing and Applications, 2014, 24(7-8), 1867-1877.
19. Wang, Y., Cai, S., Yin, M. Two Efficient Local Search Algorithms for Maximum Weight Clique Problem. In AAAI,2016, pp.805-811.
20. Wang, Y.,Yin, M., Ouyang, D., Zhang, L. A novel local search algorithm with configuration checking and scoring mechanism for the set k-covering problem. International Transactions in Operation Research, 2017, 24(26),1436-1485.
21. Cai, S., Su, K., Sattar, A. Local search with edge weighting and configuration checking heuristics for minimum vertex cover. Artificial Intelligence, 2011, 175(9-10), 1672-1696.
22. Luo, C., Cai, S., Wu, W., Su, K. Double Configuration Checking in Stochastic Local Search for Satisfiability. In AAAI, 2014,pp.2703-2709.
23. Luo, C., Cai, S., Wu, W., Jie, Z., Su, K. CCLS: an efficient local search algorithm for weighted maximum satisfiability. IEEE Transactions on Computers, 2015, 64(7), 1830-1843.
24. Gao, J., Wang, J., Yin, M. Experimental analyses on phase transitions in compiling satisfiability problems. Science China Information Sciences, 2015, 58(3), 1-11.
25. Cai, S., Su, K., Luo, C., Sattar, A. NuMVC: An efficient local search algorithm for minimum vertex cover. Journal of Artificial Intelligence Research, 2013, 46, 687-716.