Random Walk in Large Real-World Graphs for Finding Smaller Vertex Cover

Zongjie Ma, Yi Fan, Kaile Su, Chengqian Li and Abdul Sattar

***Abstract*—The problem of finding a minimum vertex cover (MinVC) in a graph is a prominent NP-hard problem of great importance in both theory and application. During recent decades, there has been much interest in finding optimal or near-optimal solutions to this problem. Many existing heuristic algorithms for MinVC are based on local search strategies. Recently, an algorithm called FastVC takes a first step towards solving the MinVC problem for large real-world graphs. However, FastVC may be trapped by local minima during the local search stage due to the lack of suitable diversification mechanisms. In this work, we design a new random walk strategy to help FastVC escape from local minima. Experiments conducted on a broad range of large real-world graphs show that our algorithm outperforms state-of-the-art algorithms on most classes of the benchmark and finds smaller vertex covers on a considerable portion of the graphs.**

1. Introduction

Many data sets can be represented as graphs, and the study of large real-world graphs, also known as complex networks [16], has become an active research agenda over recent decades. Large graphs can be found from the Network Data Repository online [12]. Some of these graphs have recently been exploited in testing parallel algorithms for Maximum Clique [14] and Coloring problems [13][1].

We are interested in the MinVC problem for large real-world graphs. Given an undirected graph $G = (V, E)$, where V is its vertex set and E is the edge set, we say a subset $S \subseteq V$ is a vertex cover if every edge in G has at least one endpoint in S. The objective of MinVC is to find a vertex cover with minimum size in a graph. MinVC is a prominent NP-hard problem [8], and algorithms for MinVC can be directly applied to solve many other combinatorial problems such as Maximum Independent Set (MIS) and Maximum Clique (MC) problems. MinVC (MIS, MC) is of great practical importance. The relevant applications include network security, scheduling, very-large-scale integration (VLSI) design, computer vision, information retrieval, signal transmission, industrial machine assignment [3], [11], and aligning DNA and protein sequences [6], [7], [10]. In this work, our focus is on a local search approach to solving MinVC for large graphs.

To redesign a local search algorithm for MinVC in large graphs, researchers may avoid those traditional techniques with the high computational cost or implement them in an approximate but efficient way. As a good example, FastVC [2] was designed by withdrawing or modifying some techniques in NuMVC [3]. Specifically, the best-picking heuristic in NuMVC is replaced by a low-complexity approximate heuristic named Best from Multiple Selection (BMS) in FastVC. Besides, FastVC withdraws the edge weighting technique in NuMVC, because the complexity of edge weighting is too high to handle large graphs.

However, FastVC lacks some suitable mechanisms for diversification, and may be trapped in a local optimum frequently during the local search stage. We believe that it needs some low-complexity diversification strategy to help the local search escape from local minima.

In this work, we design a random walk heuristic to diversify the search. Random walk provides a mechanism to effectively avoid local optima. We combined random walk with BMS to form a new heuristic, named WalkBMS. Based on WalkBMS, we propose a new algorithm called WalkVC, which is dedicated to solve the MinVC problem in large graphs.

We conduct experiments on a broad range of large real-world graphs. Experimental results show that WalkVC significantly outperforms FastVC on solution quality for **10** classes of this benchmark, and finds the same quality solutions as FastVC on the remaining **2** classes. WalkVC finds higher-quality covers on a considerable portion of the graphs.

Our further experiments are to test FastVC on a considerable portion of the graphs with a cutoff of 100,000 seconds. Experimental results show that even within such a large cutoff, FastVC does not get the same solution quality as WalkVC does with a cutoff of 1,000 seconds for these graphs. That is, our solver is *at least 100 times as efficient as* FastVC on these graphs.

Besides, we also make a comparison with an exact branch-and-bound algorithm named B&R [1]. The results show that WalkVC can obtain solutions on 25 large or hard instances within 1000 seconds, while B&R fails to return solutions for these 25 instances within 24 hours.

- *Zongjie Ma (zongjie.ma@griffithuni.edu.au), Yi Fan, Kaile Su and Abdul Sattar are with Institute for Integrated and Intelligent Systems, Griffith University, Brisbane, Australia.*
- *Chengqian Li is with Department of Computer Science, Sun Yat-sen University, China.*

1. http://www.graphrepository.com/networks.php

2. Preliminaries

2.1. Definitions and Notation

A simple undirected graph $G = (V, E)$ consists of a vertex set $V = \{v_1, v_2, ...v_n\}$ and an edge set $E \subseteq V \times V$, where each edge is a 2-element subset of V. Given an edge $e = \{u, v\}$ where $u, v \in V$, the vertices u and v are called the endpoints of edge e. Two vertices are neighbors if and only if there exists an edge between them. We define the neighborhood of v as $N(v) = \{u \in V | \{u, v\} \in E\}$. The degree of a vertex v, denoted by $d(v)$, is defined as $|N(v)|$ which is equal to the number of its neighbors.

A current candidate solution C is a set of vertices selected for covering. For a vertex $v \in C$, the *loss* of v, denoted as $loss(v)$, is defined as the number of covered edges that will become uncovered after the removal of v from C. For a vertex $v \notin C$, the *gain* of v, denoted as $gain(v)$, is defined as the number of uncovered edges that will become covered after the addition of v into C [2]. Both *loss* and *gain* are *scoring properties* of vertices. In any step, a vertex v has two possible states: inside C and outside C. We use $age(v)$ to denote the number of steps that have been performed since last time the state of v was changed.

2.2. Review of FastVC

FastVC [2] is simple and works particularly well for large graphs. FastVC proposed two low-complexity heuristics: one is used to construct a starting vertex cover, and the other is utilized to choose the removed vertex in each exchanging step. We show FastVC in Algorithm 1.

Algorithm 1: FastVC

input : a graph $G = (V, E)$, the cutoff time
output: a vertex cover of G

```
1  C ← ConstructVC() ;
2  while elapsed time < cutoff do
3      if C covers all edges then
4          C* ← C;
5          remove a vertex with minimum loss from C;
6          continue;
7      u ← BMS(C, 50);
8      remove u from C;
9      e ← a random uncovered edge;
10     v ← the endpoint of e with greater gain,
           breaking ties in favor of the older one;
11     add v into C;
12 return C*;
```

2.2.1. The Construction Stage. In the beginning, a vertex cover C is constructed by the $ConstructVC$ function, and will be used as the starting vertex cover. In detail, the $ConstructVC$ function consists of an extending phase and a shrinking phase. In the extending phase, the heuristic works as follows:

Repeat the following operations until C becomes a cover: select an uncovered edge and add the endpoint with higher degree into C.

Then in the shrinking phase, redundant vertices (vertices whose $loss$ is 0) are removed by a read-one procedure. Such a construction procedure is suitable for large graphs, since it outputs quite a good starting vertex cover typically within 1 second. Also its complexity is proved to be $O(|E|)$ [2].

2.2.2. The Local Search Stage. In the local search stage, the exchanging step of FastVC adopts the two-stage exchange framework proposed in NuMVC [3]. In the vertex-removing stage, a vertex is selected by BMS (Line 7) whose details are described as follows:

Choose k vertices randomly with replacement from C, and then return the vertex with minimum loss, breaking ties in favor of the oldest one.

Algorithm 2: BMS

input : a vertex set C, a positive integer k
output: a vertex $u \in C$

```
1 Let S be an empty vertex set;
2 for iteration ← 1 to k do
3     u ← a random vertex from C;
4     S ← S ∪ {u};
5 choose a vertex v from S with the minimum loss,
    breaking ties in favor of the oldest one;
6 return v;
```

BMS has a complexity of $O(1)$. The probability that BMS chooses a vertex whose $loss$ value is not larger than 90% vertices in C is 99.48% (when we set k=50) [2]. This means that BMS will probably return a high-quality vertex. That is, BMS approximates the minimum $loss$ removing heuristic in NuMVC [3] very well.

Then in the vertex-adding stage (Lines 9 to 11), the algorithm randomly selects an uncovered edge e, and chooses the endpoint of e with greater gain (breaking ties in favor of the older one) to add it into C. Note that after changing the state of a vertex (removing or adding), the algorithm will update the $loss$ and $gain$ values of the vertex and its neighbors.

3. WalkBMS Heuristic and WalkVC Solver

Since FastVC is designed by withdrawing or modifying some techniques in NuMVC [3], we make a comparison between them. We find that there are four diversification strategies in NuMVC including tabu [5], Configuration Checking (CC) [4], edge weighting and random selection of an uncovered edge. In contrast FastVC only exploits two diversification strategies: BMS and random selection of an uncovered edge. Thus in our opinion, there are too few diversification mechanisms in FastVC. Moreover BMS will

choose a good vertex very probably, so the diversification effect in BMS is very limited. Therefore FastVC may be trapped by local optima. So in this work, we will add a diversification strategy to help FastVC escape from local optima.

Currently there are many diversification strategies to try, such as tabu, CC, edge weighting [4], [11], vertex weighting [9], and random walk. Tabu strategies prevent local search from canceling the effects of the previous steps. Edge weighting and vertex weighting guide local search to the part of search space which is rarely explored. CC help overcome the cycling problem. Random walk allows increasing the number of unsatisfied constraints occasionally, and is successfully used in the satisfiability (SAT) problem [15].

Considering that we are solving the MinVC problem on large graphs, the complexity of the heuristics is an important issue, because the high complexity severely limits the ability of algorithms to deal with huge instances. Among the diversification strategies above, the complexity of tabu, CC and random walk is $O(1)$, while edge weighting has a complexity of $O(|E|)$ and vertex weighting's complexity is $O(|V|)$. So the complexity of edge weighting and vertex weighting are too high to handle large data sets. As to tabu and CC, our experiments did not show significant improvements. Yet after incorporating random walk into the vertex-removing stage, we found a highly significant progress. To our best knowledge, it is the first algorithm applying random walk to remove a vertex in the two-stage exchange framework. We call this new heuristic WalkBMS.

Specifically, WalkBMS combines a random walk with the BMS strategy as below:

- *With probability p, follow* BMS*;*
- *With probability $1-p$, choose a random vertex.*

Throughout this paper, the probability p is fixed in advance: we set $p = 0.6$ and $k = 50$ (the same value of k as FastVC in [2]) for all of the experiments[2]. Like FastVC, the parameter p in this study is also instance-independent.

We formalize the WalkBMS in Algorithm 3 as below.

Algorithm 3: WalkBMS

input : a vertex set C,
a probability parameter p, a positive integer k
output: a vertex $v \in C$

1 With probability p: $v \leftarrow$ BMS(C, k);
2 With probability $1-p$: $v \leftarrow$ a random vertex in C;
3 **return** v;

WalkBMS switches between the greedy mode (Line 1, BMS mode) and the diversification mode (Line 2) at a certain probability. In the greedy mode, WalkBMS exploits BMS directly to choose a vertex for removing from the current candidate solution C; In the diversification mode, WalkBMS selects a vertex randomly from C.

WalkBMS is utilized to develop a new algorithm for handling the MinVC problem in large real-world graphs directly: replace the BMS function in FastVC (Line 7, Algorithm 1) with our WalkBMS function. We call this new algorithm WalkVC.

3.1. Relationship with Other Methods

Random walk is an efficient and effective method to improve local search with a very low complexity. Also it has been successfully used in the satisfiability (SAT) problem which is an NP-complete problem. Random walk in SAT picks a variable from a random unsatisfied constraint (clause) and flip it, which provides a mechanism to escape from local minima effectively. WalkSAT [15], a SAT solver depending on random walk, is still a state-of-the-art solver on huge random 3-SAT instances. Note that the random walk proposed in this study is essentially different from those existing in the SAT or MinVC solver. Previous random walk focuses on choosing a variable (vertex) from a random unsatisfied constraint (unsatisfied clause or uncovered edge), while our random walk simply chooses a vertex from C. This is also the first time random walk is applied in the vertex-removing stage of the two-stage exchange framework.

4. Experiment Evaluation

4.1. Benchmarks

We downloaded all 139 instances[3], which were originally online[4], and then transformed to DIMACS graph format. In such a format, the size of an input file storing a graph G, is proportional to the number of edges in G. We excluded three extremely large ones, since they are out of memory for the two algorithms here. Therefore the remaining 136 instances are used for testing the solvers in our experiments. In many of these large real-world graphs there are millions of vertices and dozens of millions of edges. Recently, some of these graph data are utilized to evaluate parallel algorithms for Maximum Clique [14] and Coloring problems [13].

4.2. Experiment Setup

In the experiments, we mainly compare WalkVC with FastVC. Since FastVC is the state-of-the-art local search based algorithm on finding vertex covers in large graphs, and the similar algorithm structure between our WalkVC and FastVC helps to show the effectiveness of *random walk*. Besides, we also make a comparison with B&R [1], the state-of-the-art branch-and-bound algorithm , which helps

2. Different values of p are only used to test parameter sensitivity.

3. http://lcs.ios.ac.cn/∼caisw/Resource/realworld%20graphs.tar.gz

4. http://www.graphrepository.com/networks.php

to evaluate the absolute quality of solutions returned by WalkVC. The experiments were conducted on a cluster equipped with a number of Intel(R) Xeon(R) CPUs X5650 @2.67GHz with 8GB RAM, running Red Hat Santiago OS.

4.2.1. WalkVC and FastVC. WalkVC[5] and FastVC[6] were implemented in C++, and they were compiled by g++ 4.6.3 with the '-O3' option.

As shown in Algorithms 3, there are two parameters in WalkVC algorithms. In our experiments, the parameter p is set to 0.6, and k is set to 50 (the same as the default setting in FastVC). For FastVC, we adopt the parameter setting reported in [2].

WalkVC and FastVC are performed 10 times on each instance with a time limit of 1,000 seconds. For each algorithm on each instance, we report the minimum size ("C_{min}") and averaged size ("C_{avg}") of vertex covers found by the algorithm. To make the comparisons clearer, we report the difference ("Δ") between the minimum size of vertex cover found by WalkVC and FastVC. A positive Δ means WalkVC finds a smaller vertex cover, while a negative Δ means FastVC finds a smaller vertex cover.

4.2.2. B&R. The exact algorithm B&R[7] was compiled and run with Java 1.8.0_66. The time limit is set to 24 hours for each execution, and timeouts are denoted as '-' in related tables.

4.3. Results with FastVC

Table 1 contains all the graph instances where WalkVC and FastVC return different C_{min} or C_{avg} values.

4.3.1. Quality Improvements. Out of the 43 graphs in Table 1,

1) WalkVC finds better solutions for 24 graphs.
2) FastVC finds better solutions for 5 graphs.

4.3.2. Success Rate Improvements. As is shown in Table 1, for those **14** graphs where $\Delta = 0$, WalkVC obtains smaller C_{avg} values for **11** graphs.

4.3.3. Robustness Improvements. Over all the 12 classes of instances, compared to FastVC,

1) WalkVC returns better quality solutions in 10 classes.
2) It finds the same C_{min} and C_{avg} values in 2 classes.

4.3.4. Speed Improvements. Over half of the 24 graphs where we found smaller covers, WalkVC makes a substantially large progress. Now we show how great the progress is. We enlarged the cutoff to be 100 times as large as before (i.e., **100,000s**), and tested FastVC over such graphs. The results are shown in Table 2. Also we present the respective results of WalkVC within **1,000 seconds** in this table.

As is shown in Table 2, even within such a large cutoff, FastVC does not get the same solution quality as WalkVC does with a cutoff of 1,000 seconds for any of these 12 graphs. That is, our solver is *at least 100 times as efficient as* FastVC on these graphs.

5. https://github.com/math6068/WalkVC
6. http://lcs.ios.ac.cn/∼caisw/Code/FastVCv2015.11.zip
7. https://github.com/wata-orz/vertex_cover

TABLE 1. EXPERIMENTAL RESULTS ON LARGE REAL-WORLD GRAPHS. A POSITIVE Δ MEANS WALKVC FINDS A SMALLER VERTEX COVER, WHILE A NEGATIVE Δ MEANS FASTVC FINDS A SMALLER VERTEX COVER. FOR $\Delta \neq 0$, WE BOLD THE SMALLER VALUE OF MINIMUM SIZE (C_{min}) BETWEEN THE TWO ALGORITHMS, AND FOR $\Delta = 0$, WE BOLD THE SMALLER VALUE OF AVERAGE SIZE (C_{avg}).

Graph	$\|V\|$	$\|E\|$	FastVC $C_{min}(C_{avg})$	WalkVC $C_{min}(C_{avg})$	Δ
socfb-A-anon	3097165	23667394	**375231**(375232.8)	375232(375232.9)	-1
socfb-B-anon	2937612	20959854	303048(**303048.8**)	303048(303048.9)	0
socfb-Berkeley13	22900	852419	**17210**(17212.8)	17211(17212.6)	-1
socfb-CMU	6621	249959	4986(**4986.5**)	4986(4986.8)	0
socfb-Duke14	9885	506437	7683(7683.1)	7683(**7683**)	0
socfb-Indiana	29732	1305757	23315(23317.3)	23315(**23316.3**)	0
socfb-OR	63392	816886	36548(36549.2)	36548(**36548.1**)	0
socfb-Penn94	41536	1362220	31162(31164.8)	**31161**(31163)	1
socfb-Stanford3	11586	568309	8518(8518)	**8517**(8517.9)	1
socfb-Texas84	36364	1590651	28167(28171.4)	**28166**(28170)	1
socfb-UCLA	20453	747604	15223(15224.3)	**15222**(15223.8)	1
socfb-UConn	17206	604867	**13230**(13231.6)	13231(13231.4)	-1
socfb-UCSB37	14917	482215	11261(11263.1)	11261(**11261.8**)	0
socfb-UF	35111	1465654	27306(27309.1)	**27305**(27307.9)	1
socfb-UIllinois	30795	1264421	24091(**24092.6**)	24091(24093.9)	0
socfb-Wisconsin87	23831	835946	18383(18385.1)	18383(**18384.3**)	0
ia-infect-dublin	410	2765	293(293.5)	293(**293**)	0
inf-roadNet-CA	1957027	2760388	1001273(1001310.9)	**1001263**(1001315.1)	10
inf-roadNet-PA	1087562	1541514	555220(555242.8)	**555205**(555235.9)	15
rec-amazon	91813	125704	47606(47606)	**47605**(47605.8)	1
rt-retweet-crawl	1112702	2278852	81048(81048)	**81045**(81047.5)	3
sc-ldoor	952203	20770807	856755(856757.4)	856755(**856757.2**)	0
sc-nasasrb	54870	1311227	51244(51247.4)	**51241**(51242.2)	3
sc-pkustk11	87804	2565054	83911(83912.5)	83911(**83911.5**)	0
sc-pkustk13	94893	3260967	**89217**(89220.6)	89231(89234.5)	-14
sc-pwtk	217891	5653221	207716(207719.9)	**207712**(207716.3)	4
sc-shipsec1	140385	1707759	117318(117338.4)	**117256**(117284.1)	62
sc-shipsec5	179104	2200076	147140(147175)	**147124**(147163.3)	16
soc-buzznet	101163	2763066	30625(30625)	**30618**(30621.6)	7
soc-delicious	536108	1365961	85686(85696.4)	**85597**(85638.8)	89
soc-digg	770799	5907132	103244(103245.3)	**103243**(103244.5)	1
soc-flickr	513969	3190452	153272(153272)	**153271**(153272.1)	1
soc-FourSquare	639014	3214986	90109(90109.3)	**90108**(90108.5)	1
soc-livejournal	4033137	27933062	1869045(1869053.7)	**1869035**(1869049.1)	10
soc-pokec	1632803	22301964	843422(843434.8)	843422(**843429.1**)	0
tech-as-skitter	1694616	11094209	527185(527196)	**527177**(527199.9)	8
tech-RL-caida	190914	607610	74930(74938.9)	**74901**(74918.6)	29
scc_infect-dublin	10972	175573	9104(9104)	**9103**(9103)	1
web-arabic-2005	163598	1747269	114426(114427.2)	114426(**114427.1**)	0
web-BerkStan	12305	19500	**5384**(5384)	5385(5385)	-1
web-it-2004	509338	7178413	414671(414676.3)	414671(**414675.1**)	0
web-spam	4767	37375	2298(2298)	**2297**(2297)	1
web-wikipedia2009	1864433	4507315	648317(648321.7)	**648316**(648319.6)	1

TABLE 2. RESULTS ON THE 12 GRAPHS ON WHICH WALKVC MAKES A SUBSTANTIALLY LARGE PROGRESS.

Graph	$\|V\|$	$\|E\|$	FastVC×**100** $C_{min}(C_{avg})$	WalkVC $C_{min}(C_{avg})$	Δ
inf-roadNet-CA	1957027	2760388	1001272(1001306.3)	**1001263**(1001315.1)	9
inf-roadNet-PA	1087562	1541514	555220(555242)	**555205**(555235.9)	15
rec-amazon	91813	125704	47606(47606)	**47605**(47605.8)	1
sc-shipsec1	140385	1707759	117298(117313.8)	**117256**(117284.1)	42
sc-shipsec5	179104	2200076	147130(147171.3)	**147124**(147163.3)	6
soc-buzznet	101163	2763066	30625(30625)	**30618**(30621.6)	7
soc-delicious	536108	1365961	85685(85695.5)	**85597**(85638.8)	88
soc-digg	770799	5907132	103244(103245.3)	**103243**(103244.5)	1
tech-as-skitter	1694616	11094209	527185(527195.3)	**527177**(527199.9)	8
tech-RL-caida	190914	607610	74930(74938.9)	**74901**(74918.6)	29
scc_infect-dublin	10972	175573	9104(9104)	**9103**(9103)	1
web-spam	4767	37375	2298(2298)	**2297**(2297)	1

4.4. Results with B&R

As shown in Table 3, we do not report the instances where WalkVC and B&R return the same minimum size of vertex cover. We find that:

1) For all the 136 instances, WalkVC finds the same solutions as B&R on 99 instances. This means that WalkVC, as a local search based solver, returns optimal solutions on a considerable portion of the instances.

TABLE 3. EXPERIMENTAL RESULTS WITH B&R. WE BOLD THE SMALLER VALUE OF MINIMUM SIZE BETWEEN THE TWO ALGORITHMS. TIMEOUTS ARE DENOTED AS '-'.

Graph	$\|V\|$	$\|E\|$	B&R	WalkVC
socfb-A-anon	3097165	23667394	**375230**	375232
socfb-Berkeley13	22900	852419	-	**17211**
socfb-CMU	6621	249959	-	**4986**
socfb-Duke14	9885	506437	-	**7683**
socfb-Indiana	29732	1305757	-	**23315**
socfb-MIT	6402	251230	-	**4657**
socfb-OR	63392	816886	-	**36548**
socfb-Penn94	41536	1362220	-	**31161**
socfb-Stanford3	11586	568309	-	**8517**
socfb-Texas84	36364	1590651	-	**28166**
socfb-UCLA	20453	747604	-	**15222**
socfb-UConn	17206	604867	-	**13231**
socfb-UCSB37	14917	482215	-	**11261**
socfb-UF	35111	1465654	-	**27305**
socfb-UIllinois	30795	1264421	-	**24091**
socfb-Wisconsin87	23831	835946	-	**18383**
inf-roadNet-CA	1957027	2760388	-	**1001263**
inf-roadNet-PA	1087562	1541514	-	**555205**
rt-retweet-crawl	1112702	2278852	**81040**	81045
sc-ldoor	952203	20770807	**856754**	856755
sc-nasasrb	54870	1311227	-	**51241**
sc-pkustk13	94893	3260967	-	**89231**
sc-pwtk	217891	5653221	-	**207712**
sc-shipsec1	140385	1707759	-	**117256**
sc-shipsec5	179104	2200076	-	**147124**
soc-buzznet	101163	2763066	**30613**	30618
soc-delicious	536108	1365961	**85298**	85597
soc-digg	770799	5907132	**103234**	103243
soc-livejournal	4033137	27933062	**1868903**	1869035
soc-pokec	1632803	22301964	-	**843422**
tech-as-skitter	1694616	11094209	-	**527177**
tech-RL-caida	190914	607610	**74593**	74901
web-arabic-2005	163598	1747269	**114420**	114426
web-BerkStan	12305	19500	**5384**	5385
web-it-2004	509338	7178413	**414507**	414671
web-webbase-2001	16062	25593	**2651**	2652
web-wikipedia2009	1864433	4507315	-	**648316**

2) B&R fails to return solutions on 25 hard or large instances within 24 hours, while WalkVC obtains solutions within 1000 seconds on these instances. This illustrates that incomplete solvers are able to obtain optimal or near-optimal solutions to large or hard instances within reasonable time.
3) There are 12 instances where B&R obtains better solutions than WalkVC. For incomplete solvers, there is room for improvement on these instances.

5. Conclusions

In this work, we have developed a local search MinVC solver called WalkVC, which is based on BMS with random walk for removing vertices in the two-stage exchange framework.

Our experimental results are impressive. Firstly, WalkVC significantly outperforms FastVC on nearly all classes of large graphs. Moreover, WalkVC found smaller covers on a considerable portion of graphs. Especially over half of these graphs our solver is at least 100 times as efficient as FastVC. This shows the power of our simple random walk strategy. The results with B&R show that WalkVC could return solutions on some hard or large instance within reasonable time.

In our future work, we would like to design more efficient diversification strategies to improve the performance of our solver for the MinVC problem on large graphs.

Acknowledgment

This work is supported by ARC Grant FT0991785, NSFC grant No.61572234, NSF Grant No.61463044 and Grant No.[2014]7421 from the Joint Fund of the NSF of Guizhou province of China.

We gratefully acknowledge the support of the Griffith University eResearch Services Team and the use of the High Performance Computing Cluster "Gowonda" to complete this research.

References

[1] T. Akiba and Y. Iwata, "Branch-and-reduce exponential/fpt algorithms in practice: A case study of vertex cover." in *ALENEX*. SIAM, 2015, pp. 70–81.

[2] S. 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 2015*. Buenos Aires, Argentina: AAAI Press, 25–31 July 2015, pp. 747–753.

[3] S. Cai, K. Su, C. Luo, and A. Sattar, "Numvc: An efficient local search algorithm for minimum vertex cover," *J. Artif. Intell. Res. (JAIR)*, vol. 46, pp. 687–716, 2013.

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

[5] F. Glover, "Tabu search - part I," *INFORMS Journal on Computing*, vol. 1, no. 3, pp. 190–206, 1989.

[6] Y. Ji, X. Xu, and G. D. Stormo, "A graph theoretical approach for predicting common RNA secondary structure motifs including pseudoknots in unaligned sequences," *Bioinformatics*, vol. 20, no. 10, pp. 1603–1611, 2004.

[7] Y. Jin and J. Hao, "General swap-based multiple neighborhood tabu search for the maximum independent set problem," *Eng. Appl. of AI*, vol. 37, pp. 20–33, 2015.

[8] R. M. Karp, "Reducibility among combinatorial problems," in *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York.*, 1972, pp. 85–103.

[9] W. Pullan, "Optimisation of unweighted/weighted maximum independent sets and minimum vertex covers," *Discrete Optimization*, vol. 6, no. 2, pp. 214–219, 2009.

[10] W. J. Pullan and H. H. Hoos, "Dynamic local search for the maximum clique problem," *J. Artif. Intell. Res. (JAIR)*, vol. 25, pp. 159–185, 2006.

[11] S. Richter, M. Helmert, and C. Gretton, "A stochastic local search approach to vertex cover," in *Proc. of KI 2007: Advances in Artificial Intelligence, 30th Annual German Conference on AI*. Osnabrück, Germany: Springer-Verlag Berlin Heidelberg, 10–13 September 2007, pp. 412–426.

[12] R. Rossi and N. Ahmed, "The network data repository with interactive graph analytics and visualization," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence,*. Austin, Texas, USA: AAAI Press, 25–30 January 2015, pp. 4292–4293.

[13] R. A. Rossi and N. K. Ahmed, "Coloring large complex networks," *Social Netw. Analys. Mining*, vol. 4, no. 1, p. 228, 2014.

[14] R. A. Rossi, D. F. Gleich, A. H. Gebremedhin, and M. M. A. Patwary, "Fast maximum clique algorithms for large graphs," in *23rd International World Wide Web Conference, WWW '14*. Seoul, Republic of Korea: ACM, 7–11 April 2014, pp. 365–366.

[15] B. Selman, H. A. Kautz, and B. Cohen, "Noise strategies for improving local search," in *Proceedings of the 12th National Conference on Artificial Intelligence*. Seattle, WA, USA: AAAI Press / The MIT Press, 31 July–4 August 1994, pp. 337–343.

[16] A. L. Traud, P. J. Mucha, and M. A. Porter, "Social structure of facebook networks," *Physica A: Statistical Mechanics and its Applications*, vol. 391, no. 16, pp. 4165–4180, 2012.