Random Walk in Massive Graphs for Finding Smaller Vertex Cover

Abstract

The problem of finding a minimum vertex cover (MinVC) in a graph is a prominent NP-hard problem of great importance in both theories and applications. During last 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 strategy. Recently, an algorithm called FastVC takes a first step towards solving the MinVC problem for massive real world graphs. However, after withdrawing the edge weighting techniques, FastVC may be trapped by local minima 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 massive real world graphs show that our algorithm outperforms FastVC 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 massive real world graphs, also known as complex networks [Traud *et al.*, 2012], has become an active research agenda over recent decades. For combinatorial optimization problems like the Minimum Vertex Cover (MinVC) problem, new heuristics and algorithms should be designed to deal with massive graphs. Massive graphs can be found from the Network Data Repository online [Rossi and Ahmed, 2015]. Many of these real world graphs have millions of vertices and dozens of millions of edges. Some of these graphs have recently been exploited in testing parallel algorithms for Maximum Clique [Rossi *et al.*, 2014] and Coloring problems [Rossi and Ahmed, 2014][1]. For such graphs, most previous heuristics and algorithms do not work well due to the computational cost.

We are interested in the MinVC problem for massive 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 [Karp, 1972], 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 [Cai *et al.*, 2013; Richter *et al.*, 2007], and aligning DNA and protein sequences [Pullan and Hoos, 2006; Ji *et al.*, 2004; Jin and Hao, 2015]. The existing algorithms for MinVC include exact ones and heuristic ones. Exact algorithms may fail to return a good solution within reasonable time for massive graphs. On the other hand, although heuristic methods, such as local search, fail to confirm the optimality of the returned solutions, they are able to obtain optimal or near-optimal solutions for massive graphs within reasonable time. In this work, our focus is on local search approach to solving MinVC for massive graphs.

To redesign a local search algorithm for MinVC in massive graphs, researchers may withdraw those traditional techniques with the high computational cost or implement them in an approximate but efficient way. As a good example, FastVC [Cai, 2015] was designed by withdrawing or modifying some techniques in NuMVC [Cai *et al.*, 2013]. 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 massive 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 has been successfully used in the satisfiability (SAT) problem [Selman *et al.*, 1994], as it 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

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

algorithm called WalkVC, which is dedicated to solve the MinVC problem in massive graphs.

We conduct experiments to compare WalkVC with FastVC on a broad range of massive real world graphs. We choose FastVC as the compared algorithm for two reasons. First, to the best of our knowledge, FastVC outperforms other existing algorithms on finding vertex covers in massive graphs. Second, its similar algorithm structure as WalkVC helps to validate the effectiveness of *random walk*.

Experimental results show that WalkVC significantly outperforms FastVC on solution quality for **9** classes of this benchmark, and finds the same quality solutions as FastVC on the remaining **3** 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. Actually in the MinVC research literature, it is rare to improve the solution quality [Cai, 2015]. The existing MinVC algorithms often obtain the solutions with the same quality, and they focus on comparing the success rate of finding a solution of such a quality.

The rest of the paper is organized as follows. Section 2 presents some background works, and the algorithm of FastVC. Then, we describe our novel heuristic and algorithm for the MinVC problem in massive graphs in section 3. In section 4, we carry out extensive experiments to evaluate WalkVC. Finally, we conclude our work in section 5.

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. An edge $e \in E$ is covered by a vertex set $S \subseteq V$ if at least one endpoint of e is present in S; otherwise, e is uncovered by S.

Given a graph $G = (V, E)$, we use $\bar{G} = (V, \bar{E})$ to denote the complementary graph where $\bar{E} = \{(u, v) | (u, v) \notin E\}$. Then given a subset $S \subseteq V$, the following three statements are equivalent [Wu and Hao, 2015]: S is a vertex cover in G, $V \backslash S$ is an independent set in G and $V \backslash S$ is a clique in $\bar{G}$. Thus to find a maximum independent set of a graph G, we can firstly find a minimum vertex cover C_{min} of G and then return $V \backslash C_{min}$. Likewise, to find a maximum clique of a graph G, we can find a minimum vertex cover $\bar{C}_{min}$ of the complementary graph $\bar{G}$ and then return $V \backslash \bar{C}_{min}$.

2.2 Local Search Strategy for MinVC

Since MinVC is important to many real-world applications, a number of algorithms have been proposed in the past. There are roughly two categories of practical MinVC (MIS, MC) algorithms: branch-and-bound algorithms and incomplete algorithms.

The branch-and-bound algorithms are able to confirm the optimality of the returned solutions [Segundo *et al.*, 2011; Li and Quan, 2010; Tomita and Kameda, 2007; Östergård, 2002; Carraghan and Pardalos, 1990], and are applicable to instances with limited sizes. However, with the increase of problem size, branch-and-bound methods may become ineffective for large and hard instances and fail to return a solution within reasonable time. On the other hand, although incomplete methods cannot confirm the optimality of the returned solutions, they are able to obtain optimal or near-optimal solutions to large and hard instances within reasonable cutoff [Cai *et al.*, 2011]. Among the incomplete approaches, local search is very popular, such as [Cai *et al.*, 2013; 2011; Richter *et al.*, 2007] for MinVC, [Andrade *et al.*, 2012; Barbosa and Campos, 2004; Busygin *et al.*, 2002] for MIS, and [Grosso *et al.*, 2008; Katayama *et al.*, 2005] for MC.

Usually local search for MinVC works by iteratively solving its decision version: given a positive integer k, searching for a k-sized vertex cover. The current candidate solution is denoted as C, which is a set of vertices selected for covering. A general framework of local search for MinVC is shown in Algorithm 1, as described below.

Algorithm 1: Local Search Framework for MinVC

```
1 construct $C$ until it becomes a vertex cover;
2 while not reach terminate condition do
3     if $C$ covers all edges then
4         $C^* \leftarrow C$;
5         remove a vertex from $C$;
6     execute an exchanging step;
7 return $C^*$
```

Algorithm 1 consists of two stages: a construction stage (Line 1) and a local search stage (Lines 2 to 6). At the first stage, a vertex cover is constructed usually based on a greedy heuristic. Throughout this paper, such a vertex cover is also called the *starting vertex cover*. At the local search stage, each time the algorithm finds out a k-sized cover (Line 3), it removes a vertex from C (Line 5) and continues to search for a (k-1)-sized cover, until reaching some termination condition (Line 2).

The move to a neighboring candidate solution is actually an exchange of two vertices (Line 6): remove a vertex u from C and add a vertex v into C. This step is also called an exchanging step. Thus the local search moves step-by-step in the search space to find a better vertex cover. After it terminates, the algorithm outputs the smallest vertex cover that has been found.

As mentioned above, the 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 [Cai, 2015]. 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.

2.3 Review of FastVC

FastVC [Cai, 2015] is simple and works particularly well for massive 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 2.

Algorithm 2: FastVC

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

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 massive graphs, since it outputs quite a good starting vertex cover typically within 1 second. Also its complexity is proved to be $O(|E|)$ [Cai, 2015].

The Local Search Stage

In the local search stage, the exchanging step of FastVC adopts the two-stage exchange framework proposed in NuMVC [Cai *et al.*, 2013]. In the vertex-removing stage, a vertex is selected by BMS (Line 7) whose details are described as follows:

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

Algorithm 3: 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      $u \leftarrow$ a random vertex from $C$;
4      $S \leftarrow S \cup \{u\}$;
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% [Cai, 2015]. This means that BMS will probably return a high-quality vertex. That is, BMS approximates the minimum $loss$ removing heuristic in NuMVC [Cai *et al.*, 2013] 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

3.1 WalkBMS Heuristic and WalkVC Solver

Since FastVC is designed by withdrawing or modifying some techniques in NuMVC [Cai *et al.*, 2013], we make a comparison between them. We find that there are four diversification strategies in NuMVC including tabu [Glover, 1989], Configuration Checking (CC) [Cai *et al.*, 2011], 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 of diversification strategies to try, such as tabu, CC, edge weighting [Richter *et al.*, 2007; Cai *et al.*, 2011], vertex weighting [Pullan, 2009], 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 [Selman *et al.*, 1994].

Considering that we are solving the MinVC problem on massive 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 massive 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 [Cai, 2015]) for all of the experiments[2]. Like FastVC, the parameter p in this study is also instance-independent.

We formalize the WalkBMS in Algorithm 4 as below.

Algorithm 4: 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 massive real-world graphs directly: replace the BMS function in FastVC (Line 7, Algorithm 2) with our WalkBMS function. We call this new algorithm WalkVC.

3.2 Comparing Complexity between WalkBMS and BMS

In Algorithm 4, when setting $p = 1$, WalkBMS can be regarded as BMS, and they have the same complexity. Otherwise in Line 2, WalkBMS only chooses *one* random vertex from C at a certain probability, while BMS always randomly chooses k vertices from C with a comparison between them. This means that at this certain probability the time consumed in Line 2 of Algorithm 4 is equivalent to $1/k$ of BMS (k = 50 in FastVC), regardless of the time of comparison between k vertices in BMS.

To be more specific, we focus on comparing the average number of iterations in BMS and WalkBMS heuristic. The average number of iterations in BMS is k, while the number is $pk + (1-p)$ in WalkBMS. According to the recommended parameter in [Cai, 2015] and the fixed parameter in this study, that is, $k = 50$ and $p = 0.6$, we can calculate that the number of iterations in BMS is about 1.6 times as many as that in WalkBMS.

3.3 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 [Selman *et al.*, 1994], 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

In this section, we carry out extensive experiments to evaluate WalkVC on a wide range of real-world massive graphs, and make a comparison with the state-of-the-art local search MinVC algorithm FastVC.

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 real-world massive 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 [Rossi *et al.*, 2014] and Coloring problems [Rossi and Ahmed, 2014].

The graphs used in our experiments can be divided into 12 classes: biological networks, collaboration networks, facebook networks, interaction networks, infrastructure networks, amazon recommend networks, retweet networks, scientific computation networks, social networks, technological networks, web link networks, and temporal reachability networks.

[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

4.2 Experiment Setup

In the experiments, we compare WalkVC with FastVC. There are two reasons why we choose FastVC for comparisons. To the best of our knowledge, FastVC obtains a better performance than other existing algorithms on finding vertex covers in massive graphs. Besides, the similar algorithm structure between our WalkVC and FastVC helps to show the effectiveness of *random walk*.

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

As shown in Algorithms 4, 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 [Cai, 2015].

All the algorithms 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.

4.3 Results with FastVC

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

Quality Improvements

Out of the 42 graphs in Table 1,

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

Success Rate Improvements

As is shown in Table 1, for those **13** graphs where $\Delta = 0$, WalkVC obtains smaller C_{avg} values for **10** graphs.

Robustness Improvements

Over all the 12 classes of instances, compared to FastVC,

1. WalkVC returns better quality solutions in 9 classes.
2. It finds the same C_{min} and C_{avg} values in 3 classes.

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.

[5]http://lcs.ios.ac.cn/∼caisw/Code/FastVCv2015.11.zip

Table 1: Experimental results on real-world massive 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
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

That is, our solver is *at least 100 times as efficient as* FastVC on these graphs.

4.4 Performance with different parameter values

Table 3 shows the performance of WalkVC with different parameter settings for p ranging from 0.1 to 0.9, while k is fixed to 50 as FastVC.

1. For most settings, WalkVC wins more than FastVC.

Table 3: Experimental results on different values of p over 10 runs. $\#win$ means the number of graphs where WalkVC finds better quality covers, and the meaning of $\#lose$ is analogous. $\bar{\Delta}$ means the average Δ on all 136 graphs

p	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
$\#win/lose$	15/26	15/20	16/18	16/12	19/8	24/5	19/9	18/11	14/8
$\bar{\Delta}$	5.8	6.4	4.6	3.4	2.5	1.8	1	0.2	0.2

2. For all the settings, WalkVC finds better quality solutions than FastVC on average.

4.5 Long-time Performance of FastVC and WalkVC

To analyze the the long-time performance of FastVC and WalkVC, we test them with a cutoff of 100,000 seconds. The cover-size currently found ($|C^*|$) and respective time stamp are recorded on average over 10 runs. On huge graphs we observed similar curves, so for the sake of space we only present the curves from three large graphs in different classes (See Figure 1). We find that:

1. The best solutions are updated frequently within 1,000 seconds.
2. Very limited quality improvements can be observed after 1,000 seconds.
3. Often, the curve of WalkVC is at the down-left side of that of FastVC, i.e., no matter which cutoff we use WalkVC always outperforms FastVC.

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.

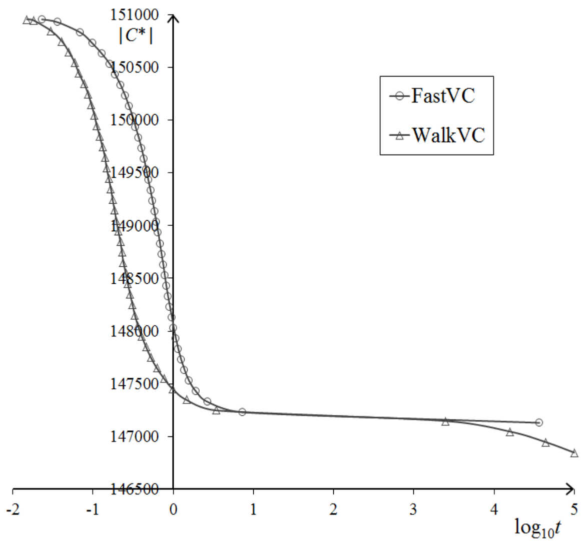
Our experimental results are impressive. Firstly, WalkVC significantly outperforms FastVC on nearly all classes of massive graphs. Moreover, WalkVC found smaller covers on a considerable portion of graphs. Especially over half of these graphs our solver is at least 100 times as efficient as FastVC. This shows the power of our simple random walk strategy.

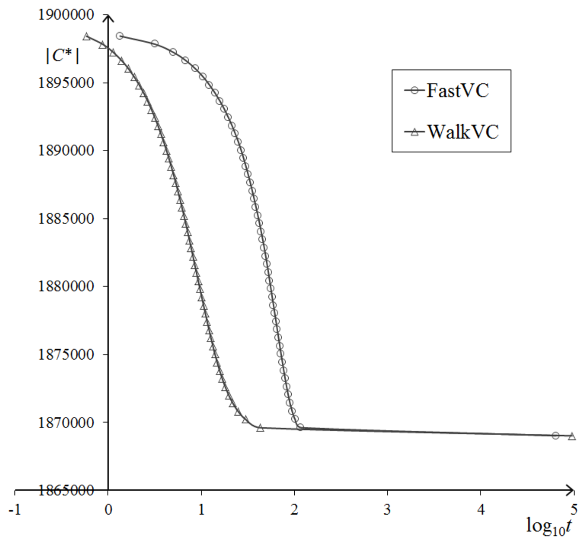
In our future work, we would like to design more efficient diversification strategies to improve the performance of our solver for the MinVC problem in massive graphs.

References

[Andrade *et al.*, 2012] Diogo Vieira Andrade, Mauricio G. C. Resende, and Renato Fonseca F. Werneck. Fast local search for the maximum independent set problem. *J. Heuristics*, 18(4):525–547, 2012.

[Barbosa and Campos, 2004] Valmir C. Barbosa and Luciana C. D. Campos. A novel evolutionary formulation of the maximum independent set problem. *J. Comb. Optim.*, 8(4):419–437, 2004.





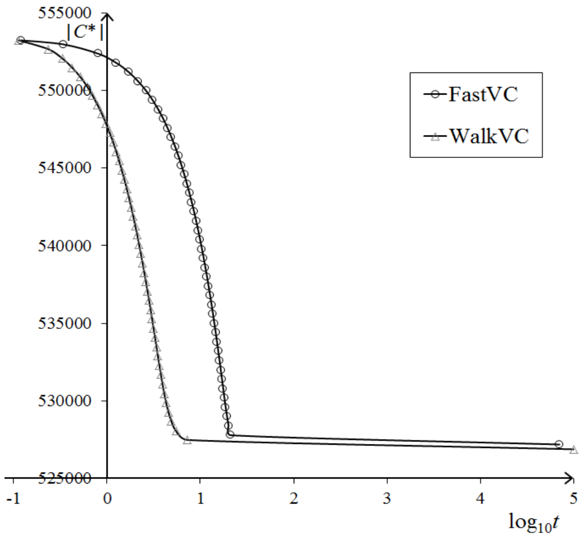


Figure 1: Long-time performance of instances *sc-shipsec5*, *soc-livejournal*, *tech-as-skitter* (from top to bottom, respectively).

[Busygin *et al.*, 2002] Stanislav Busygin, Sergiy Butenko, and Panos M. Pardalos. A heuristic for the maximum independent set problem based on optimization of a quadratic over a sphere. *J. Comb. Optim.*, 6(3):287–297, 2002.

[Cai *et al.*, 2011] Shaowei Cai, Kaile Su, and Abdul Sattar. Local search with edge weighting and configuration checking heuristics for minimum vertex cover. *Artif. Intell.*, 175(9-10):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. *J. Artif. Intell. Res. (JAIR)*, 46:687–716, 2013.

[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 2015*, pages 747–753, Buenos Aires, Argentina, 25–31 July 2015. AAAI Press.

[Carraghan and Pardalos, 1990] Randy Carraghan and Panos M Pardalos. An exact algorithm for the maximum clique problem. *Operations Research Letters*, 9(6):375–382, 1990.

[Glover, 1989] Fred Glover. Tabu search - part I. *INFORMS Journal on Computing*, 1(3):190–206, 1989.

[Grosso *et al.*, 2008] Andrea Grosso, Marco Locatelli, and Wayne J. Pullan. Simple ingredients leading to very efficient heuristics for the maximum clique problem. *J. Heuristics*, 14(6):587–612, 2008.

[Ji *et al.*, 2004] Yongmei Ji, Xing Xu, and Gary D. Stormo. A graph theoretical approach for predicting common RNA secondary structure motifs including pseudoknots in unaligned sequences. *Bioinformatics*, 20(10):1603–1611, 2004.

[Jin and Hao, 2015] Yan Jin and Jin-Kao Hao. General swap-based multiple neighborhood tabu search for the maximum independent set problem. *Eng. Appl. of AI*, 37:20–33, 2015.

[Karp, 1972] Richard 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.*, pages 85–103, 1972.

[Katayama *et al.*, 2005] Kengo Katayama, Akihiro Hamamoto, and Hiroyuki Narihisa. An effective local search for the maximum clique problem. *Inf. Process. Lett.*, 95(5):503–511, 2005.

[Li and Quan, 2010] Chu Min Li and Zhe Quan. An efficient branch-and-bound algorithm based on maxsat for the maximum clique problem. In *Proc of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010*, volume 10, pages 128–133, Atlanta, Georgia, USA, 11–15 July 2010. AAAI Press.

[Östergård, 2002] Patric RJ Östergård. A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, 120(1):197–207, 2002.

[Pullan and Hoos, 2006] Wayne J. Pullan and Holger H. Hoos. Dynamic local search for the maximum clique problem. *J. Artif. Intell. Res. (JAIR)*, 25:159–185, 2006.

[Pullan, 2009] Wayne Pullan. Optimisation of unweighted/weighted maximum independent sets and minimum vertex covers. *Discrete Optimization*, 6(2):214–219, 2009.

[Richter *et al.*, 2007] Silvia Richter, Malte Helmert, and Charles Gretton. A stochastic local search approach to vertex cover. In *Proc. of KI 2007: Advances in Artificial Intelligence, 30th Annual German Conference on AI*, pages 412–426, Osnabrück, Germany, 10–13 September 2007. Springer-Verlag Berlin Heidelberg.

[Rossi and Ahmed, 2014] Ryan A. Rossi and Nesreen K. Ahmed. Coloring large complex networks. *Social Netw. Analys. Mining*, 4(1):228, 2014.

[Rossi and Ahmed, 2015] Ryan Rossi and Nesreen Ahmed. The network data repository with interactive graph analytics and visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence,*, pages 4292–4293, Austin, Texas, USA, 25–30 January 2015. AAAI Press.

[Rossi *et al.*, 2014] Ryan A. Rossi, David F. Gleich, Assefaw Hadish Gebremedhin, and Md. Mostofa Ali Patwary. Fast maximum clique algorithms for large graphs. In *23rd International World Wide Web Conference, WWW '14*, pages 365–366, Seoul, Republic of Korea, 7–11 April 2014. ACM.

[Segundo *et al.*, 2011] Pablo San Segundo, Diego Rodríguez-Losada, and Agustín Jiménez. An exact bit-parallel algorithm for the maximum clique problem. *Computers & OR*, 38(2):571–581, 2011.

[Selman *et al.*, 1994] Bart Selman, Henry A. Kautz, and Bram Cohen. Noise strategies for improving local search. In *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 337–343, Seattle, WA, USA, 31 July–4 August 1994. AAAI Press / The MIT Press.

[Tomita and Kameda, 2007] Etsuji Tomita and Toshikatsu Kameda. An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. *Journal of Global Optimization*, 37(1):95–111, 2007.

[Traud *et al.*, 2012] Amanda L Traud, Peter J Mucha, and Mason A Porter. Social structure of facebook networks. *Physica A: Statistical Mechanics and its Applications*, 391(16):4165–4180, 2012.

[Wu and Hao, 2015] Qinghua Wu and Jin-Kao Hao. A review on algorithms for maximum clique problems. *European Journal of Operational Research*, 242(3):693–709, 2015.