Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)

Balance between Complexity and Quality: Local Search for Minimum Vertex Cover in Massive Graphs

Shaowei Cai
State Key Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences, Beijing, China
shaoweicai.cs@gmail.com

Abstract

The problem of finding a minimum vertex cover (MinVC) in a graph is a well known NP-hard problem with important applications. There has been much interest in developing heuristic algorithms for finding a "good" vertex cover in graphs. In practice, most heuristic algorithms for MinVC are based on the local search method. Previously, local search algorithms for MinVC have focused on solving academic benchmarks where the graphs are of relatively small size, and they are not suitable for solving massive graphs as they usually have high-complexity heuristics. In this paper, we propose a simple and fast local search algorithm called FastVC for solving MinVC in massive graphs, which is based on two low-complexity heuristics. Experimental results on a broad range of real world massive graphs show that FastVC finds much better vertex covers (and thus also independent sets) than state of the art local search algorithms for MinVC.

1 Introduction

The rapid growth of the Internet, the widespread deployment of sensors and scientific advances have resulted in more and more massive data sets. This has brought a series of computational challenges, as existing algorithms usually become ineffective on massive data sets, and for most problems we need to develop new algorithms. Many data sets can be modeled as graphs, and the study of massive real world graphs, also called complex networks, grew enormously in last decade.

A *vertex cover* of a graph is a subset of the vertices which contains at least one of the two endpoints of each edge. Alternatively, a vertex cover is a set of vertices whose removal completely disconnects a graph. The MinVC problem is to find the minimum sized vertex cover in a graph. MinVC is a prominent combinatorial optimization problem with important applications, including network security, industrial machine assignment and applications in sensor networks such as monitoring link failures, facility location and data aggregation [Kavalci *et al.*, 2014]. It is also closely related to the Maximum Independent Set (MaxIS) problem, which has applications in social networks, pattern recognition, molecular biology and economics [Jin and Hao, 2015].

MinVC is a classical NP-hard problem which remains intractable even for cubic graphs and planar graphs with maximum degree at most three [Garey and Johnson, 1979]. Furthermore, it is NP-hard to approximate MinVC within any factor smaller than 1.3606 [Dinur and Safra, 2005], although one can achieve an approximation ratio of $2 - o(1)$ [Karakostas, 2005].

1.1 Previous Heuristics and Motivations

Due to its NP-hardness, the research of MinVC solving is concentrated on heuristic algorithms for finding a "good" vertex cover in reasonable time. One of the most popular and efficient heuristic approaches is local search, and there has been considerable interest in local search algorithms for MinVC in last decade, e.g., [Shyu *et al.*, 2004; Richter *et al.*, 2007; Andrade *et al.*, 2008; Pullan, 2009; Cai *et al.*, 2010; 2011; 2013]. Especially, a recent algorithm called NuMVC [Cai *et al.*, 2013], which clearly dominates other local search MinVC algorithms, makes a significant improvement in MinVC solving.

Previous local search algorithms for MinVC are mainly evaluated on standard benchmarks from the academy community, particularly the DIMACS and BHOSLIB benchmarks [Richter *et al.*, 2007; Andrade *et al.*, 2008; Pullan, 2009; Cai *et al.*, 2010; 2011; 2013]. To improve the performance of the algorithms on these benchmarks, researchers have proposed sophisticated heuristics for local search for MinVC. Recent heuristics include max-gain vertex pair selection [Richter *et al.*, 2007], edge weighting [Richter *et al.*, 2007; Cai *et al.*, 2010], k-improvement (also called $(k-1, k)$ swap) [Andrade *et al.*, 2008], configuration checking [Cai *et al.*, 2011], minimum loss removing and two-stage exchange [Cai *et al.*, 2013]. Most of the previous heuristics do not have sufficiently small complexity. Because the benchmark graphs used for testing previous algorithms are not large (usually with less than five thousand vertices), the complexity of heuristics did not show an obvious impact on the performance. However, for massive graphs where the size is much larger (e.g., with millions of vertices), the high complexity severely limits the ability of algorithms to handle these data sets.

Massive graphs call for new heuristics and algorithms. However, there is little (if any) work being done on local search algorithms for massive graphs. In particular, we are not aware of any research on local search for MinVC on massive graphs. In this work, we make a first step and provide key insights in this direction. In our opinion, more attention needs to be applied to the time complexity of heuristics when designing algorithms for solving massive graphs. The key issue is on making a good balance between the time efficiency and the guidance effectiveness of heuristics.

1.2 Contributions and Paper Organization

In this paper, we design and implement a new local search algorithm for MinVC called FastVC, which is dedicated to solving massive graphs. The main principle is to use low-complexity approximate heuristics rather than those accurate heuristics with high complexity. To achieve good performance on massive graphs, we propose two new heuristics, which form the two key components of FastVC.

Firstly, we propose a fast heuristic for constructing a vertex cover. The obtained vertex cover will be used as the initialized candidate solution for local search. The heuristic considers an edge in each iteration, and if the edge is uncovered, it picks the endpoint with the higher degree to cover it. After a vertex cover is obtained, redundant vertices are removed by a read-one procedure on the vertex cover. This heuristic has a complexity of $O(|E|)$, compared to $O(|V|^2)$ for the greedy construction heuristic used in previous algorithms.

We also propose a cost-effective heuristic for choosing the vertex for removing in each step. The heuristic is named Best from Multiple Selections (BMS), which approximates the minimum loss removing heuristic [Cai *et al.*, 2013] very well and lowers the complexity from $O(|V|)$ to $O(1)$.

We carry out experiments to compare FastVC with NuMVC on a broad range of massive real world graphs. Experimental results show that FastVC finds significantly better quality vertex covers (and thus also independent sets) than NuMVC on most instances. Note that this improvement is **remarkable**, as it rarely happens in the literature to find a better solution. Previous MinVC algorithms often obtain the same quality solutions and they focus on comparing the success rate of finding a solution of such a quality.

In the next section, we introduce some necessary background knowledge. Then, we describe the FastVC algorithm in Section 3, and present two key functions based on the two new heuristics and provide some theoretical results in Section 4. Experimental evaluations of FastVC are presented in Section 5. Finally, we give some concluding remarks.

2 Preliminaries

2.1 Basic Definitions and Notation

An undirected graph $G = (V, E)$ consists of a vertex set V and an edge set E where each edge is a 2-element subset of V. For an edge $e = \{u, v\}$, we say that vertices u and v are the *endpoints* of edge e. Two vertices are neighbors if and only if they both belong to some edge. The neighborhood of a vertex v is denoted as $N(v) = \{u \in V | \{u, v\} \in E\}$, and the closed neighborhood as $N[v] = \{v\} \cup N(v)$. The *degree* of a vertex v is defined as $deg(v) = |N(v)|$. An edge $e \in E$ is *covered* by a vertex set S if at least one endpoint of e belongs to S, and otherwise it is *uncovered*.

For an undirected graph $G = (V, E)$, a *vertex cover* of a graph is a subset of V which contains at least one of the two endpoints of each edge, and an *independent set* is a subset of V where no two vertices are neighbors. A vertex set S is a vertex cover of G if and only if $V \setminus S$ is an independent set of G. We are concerned in this paper with the problem of finding a vertex cover as small as possible (MinVC). Equivalently, this problem can be viewed as seeking for an independent set as large as possible, which also has important applications.

2.2 Preliminaries of Local Search for MinVC

One popular way to solve the MinVC problem is based on iteratively solving its decision version — given a positive integer number k, searching for a k-sized vertex cover. The general scheme is as follows: In the beginning, a vertex cover is constructed; whenever the algorithm finds a vertex cover of k vertices, one vertex is removed from the vertex cover[1], and the algorithm starts from the resulting vertex set to search for a vertex set of $k - 1$ vertices that covers all edges (i.e., a vertex cover of $k - 1$ vertices) by performing local search.

We denote the current candidate solution as C, which is a vertex set containing the 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 would become uncovered by removing 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 would become covered by adding v into C. Both $loss$ and $gain$ are *scoring properties* of vertices. The age of a vertex is the number of steps since its state was last changed.

For local search MinVC algorithms based on iteratively solving the decision problem, each search step consists of exchanging a pair of vertices: a vertex $u \in C$ is removed from C, and a vertex $v \notin C$ is put into C. Such a step is called an exchanging step. In the literature, there are two ways to perform an exchanging step. The first one is adopted by algorithms before NuMVC, which chooses a vertex pair from candidate vertex pairs, and then exchanges them and updates scoring properties accordingly. The second method, proposed in NuMVC and named two-stage exchange, works in a "separate" fashion: it first chooses a vertex $u \in C$ and removes it, and updates scoring properties accordingly, and then chooses a vertex $v \notin C$ and adds it, and updates scoring properties accordingly.

3 FastVC: the Top-Level Algorithm

This section describes the FastVC algorithm on a top level. Details of important functions in FastVC and further analysis will be presented in the next section.

FastVC solves the MinVC problem by iteratively solving its decision version. For the exchanging step in local search, FastVC adopts the two-stage exchange framework, as it has lower complexity than the alternative paradigm based on vertex pair exchange. Indeed, as shown in [Cai *et al.*, 2013], thanks to the two-stage exchange framework, NuMVC

[1]if after removing one vertex, the vertex set remains a vertex cover, then more vertices are removed until it is not a vertex cover.

Algorithm 1: FastVC (G, *cutoff*)

```
Input: graph $G = (V, E)$, the cutoff time
Output: vertex cover of $G$
1  $C := ConstructVC()$;
2  $gain(v) := 0$ for each vertex $v \notin C$;
3  while elapsed time < cutoff do
4      if $C$ covers all edges then
5          $C^* := C$;
6          remove a vertex with minimum $loss$ from $C$;
7          continue;
8      $u := ChooseRmVertex(C)$;
9      $C := C \backslash \{u\}$;
10     $e :=$ a random uncovered edge;
11     $v :=$ the endpoint of $e$ with greater $gain$, breaking ties in
       favor of the older one;
12     $C := C \cup \{v\}$;
13 return $C^*$;
```

performs several times more steps per second than other local search MinVC algorithms, which is a main factor of its good performance.

The top-level algorithm of FastVC is outlined in Algorithm 1, as described below. In the beginning, a vertex cover is constructed by the $ConstructVC$ function, which is taken as the initialized candidate solution C for the algorithm. The $loss$ values of vertices in C are calculated in the $ConstructVC$ function. For vertices outside C, their $gain$ values are set to 0, as at this point all edges are covered by C and adding any vertex into C would not increase the number of covered edges.

Now we introduce the exchanging step in FastVC. At each step, the algorithm first chooses a vertex in $u \in C$ to remove, which is accomplished by the $ChooseRmVertex$ function. Then, the algorithm picks a random uncovered edge e, and chooses one of e's endpoints with the greater $gain$ and adds it into C, breaking ties in favor of the older one. Note that along with removing or adding a vertex, the $loss$ and $gain$ values of the vertex and its neighbors are updated accordingly.

4 Main Functions and Analysis

Two important components of the FastVC algorithm are the $ConstructVC$ function and the $ChooseRmVertex$ function. In this section, we present these two functions in detail, and provide some theoretical analysis on them.

4.1 Fast Construction of Vertex Cover

Local search algorithms for MinVC based on iteratively solving the decision problem starts with a vertex cover, and we call it the starting vertex cover. A small starting vertex cover can save the subsequent local search from too much unnecessary search before beginning the search for a good solution. A balance much be struck between the quality of the starting vertex cover and the time consumed in constructing it. Otherwise, the resulting algorithm may be inefficient in practice. For convenience of discussions on complexity, let us denote $n = |V|$, $m = |E|$.

Previous Construction Heuristic

Previous local search algorithms for MinVC construct the starting vertex cover using an intuitive greedy procedure [Papadimitrious and Steiglitz, 1982] which works as follows:

Repeat the following operations until C becomes a vertex cover: select a vertex $v \notin C$ with the maximum gain to add into C, breaking ties randomly.

The number of iterations of this procedure equals the size of the obtained vertex cover C, and is denoted as ℓ. We analyze the worst case complexity for two implementations of the above algorithm as follows:

A straight-forward implementation is to scan the vertex set V in each iteration in order to find the objective vertex, which has a complexity of $\Theta(n)$ for each iteration. Therefore, the complexity is $\Theta(\ell \cdot n) = O(n^2)$.

A more "clever" implementation is to maintain the C set and also a set of vertices not in C, which is denoted as H. In each iteration, we scan the H set to find a vertex with the maximum $gain$. To be precise, we use C_i and H_i to denote the C set and H set at the beginning of the i^{th} iteration. We have $|C_i| = i - 1$ and $|H_i| = n - |C_i| = n - (i - 1)$. Thus, $\sum_{i=1}^{\ell} |H_i| = \sum_{i=1}^{\ell}(n - (i - 1)) = \frac{1}{2}\ell(2n + 1 - \ell)$. Since $1 \leq \ell \leq n \Rightarrow \frac{1}{2}\ell(n + 1) \leq \frac{1}{2}\ell(2n + 1 - \ell) \leq \ell n$, we have $\sum_{i=1}^{\ell} |H_i| = \Theta(\ell \cdot n)$. Therefore, the complexity of this implementation is $\sum_{i=1}^{\ell} |H_i| = \Theta(\ell \cdot n) = O(n^2)$.

Although using "clever" implementations can accelerate the heuristic, the complexity remains square. We will see from the experiment results in Section 5.3 that, a square complexity is too high for massive graphs and makes the algorithms inefficient so that they may fail to provide a vertex cover within reasonable time (like 1000 seconds).

The ConstructVC Procedure

Our first step to build a fast local search algorithm for MinVC is to propose a fast vertex cover construction procedure, which is the $ConstructVC$ procedure in FastVC.

The proposed procedure (Algorithm 2) consists of an extending phase and a shrinking phase.

The extending phase: Starting with an empty set C, the algorithm extends C by checking and covering an edge in each iteration: if the considered edge is uncovered, the endpoint with a higher degree is added into C. Obviously, we obtain a vertex cover at the end of the extending phase.

The shrinking phase: First, we calculate the $loss$ values of vertices in C; then, we scan the C set and if a vertex $v \in C$ has a $loss$ value of 0, it is removed, and $loss$ values of its neighbors are updated accordingly.

Theorem 1. *The vertex set returned by the $ConstructVC$ procedure is a minimal vertex cover.*[2]

Proof. Let us denote the vertex cover produced by the extending phase as $C = \{v_{i_1}, v_{i_2}, v_{i_3}, ..., v_{i_\ell}\}$, which may be modified in the shrinking phase. In the j^{th} iteration of the shrinking phase, we consider vertex v_{i_j}.

(a) We first prove that the vertex set returned by the procedure is a vertex cover. At the beginning of the shrinking

[2]A vertex cover is minimal if taking any vertex out of it would make it not a vertex cover.

phase, C is a vertex cover. Suppose at the j^{th} iteration of the shrinking phase, C is a vertex cover, we will prove that at the $(j+1)^{th}$ iteration, C is a vertex cover. If $loss(v_{i_{j+1}}) > 0$, then C does not change; if $loss(v_{i_{j+1}}) = 0$, then $v_{i_{j+1}}$ is removed, but according to the definition of $loss$, removing such a vertex would not generate any new uncovered edge. Hence, C is a vertex cover after the shrinking phase.

(b) Now we prove that the vertex cover C after the shrinking phase is minimal. Suppose after the shrinking phase, there exists a vertex in C whose removal keeps C a vertex cover. Without the loss of generality, let this vertex be v_{i_j}, the one considered at the j^{th} iteration of the shrinking phase. From the assumption, we have $loss(v_{i_j}) = 0$ at the end of the shrinking phase. Notice that during the shrinking phase, the $loss$ value of any vertex in C does not decrease (according to Lemma 1). Thus, the value of $loss(v_{i_j})$ at the j^{th} iteration is at most 0, but $loss$ values are non-negative, so it is 0. Therefore, v_{i_j} would have been removed at the j^{th} iteration. This completes the proof by contradiction. □

Algorithm 2: ConstructVC (G)

```
   Input: graph $G = (V, E)$
   Output: vertex cover of $G$
1  $C := \emptyset$;
2  //extend $C$ to cover all edges
3  foreach $e \in E$ do
4      if $e$ is uncovered then
5          add the endpoint of $e$ with higher degree into $C$;
6  //calculate $loss$ of vertices in $C$
7  $loss(v) := 0$ for each $v \in C$;
8  foreach $e \in E$ do
9      if only one endpoint of $e$ belongs to $C$ then
10         for the endpoint $v \in C$, $loss(v)$++;
11 //remove redundant vertices
12 foreach $v \in C$ do
13     if $loss(v) = 0$ then
14         $C := C \backslash \{v\}$, update $loss$ of vertices in $N(v)$;
15 return $C$;
```

Lemma 1. *For any iteration of the shrinking phase, the execution of the iteration does not decrease the $loss$ value of any vertex in C.*

Proof. Consider any iteration in the shrinking phase. For simplicity, we use v to denote the vertex under consideration.

If $loss(v) > 0$, the iteration does nothing and thus the lemma holds.

Now we consider the case $loss(v) = 0$. This indicates all vertices in $N(v)$ belong to C. Otherwise, if there is a vertex $v' \in N(v)$ such that $v' \notin C$, then removing v would would make the edge $e = \{v, v'\}$ uncovered, which means $loss(v)$ is at least one. So, in the case $loss(v) = 0$, v is removed, and along with that, the $loss$ value of each vertex $v' \in N(v)$ is increased by one, as after this iteration, the removal of v' would make the edge $e = \{v, v'\}$ from covered to uncovered. For vertices not in $N(v)$, their $loss$ values do not change. □

Complexity of ConstructVC: The ConstructVC procedure can be divided into three parts: the first part (lines 3-5) performs the extending phase, the second part (lines 7-10) initializes the $loss$ values, while the last one (lines 12-14) removes redundant vertices.

Let C^+ denotes the vertex cover obtained by the extending phase. It is clear that the complexity of the extending phase is $O(m)$. For the second part, the complexity is $O(|C^+| + m)$. Since at most one vertex is added in each iteration of the extending phase, we have $|C^+| \leq m$, and thus the complexity for the second part is $O(m)$. For the last part, the complexity depends on the total number of updating operations of $loss$ values, which is calculated as $\sum_{v \in C^+} deg(v) < \sum_{v \in V} deg(v) = 2m$. Therefore, the $ConstructVC$ procedure has a complexity of $O(m)$, which is lower than $O(n^2)$ in most cases. Especially, most massive real world graphs are sparse graphs [Barabási and Albert, 1999; Eubank *et al.*, 2004; Lu and Chung, 2006], and heuristics with $O(m)$ complexity are much faster than those with $O(n^2)$ for such graphs.

4.2 Best from Multiple Selections (BMS)

A critical function of FastVC is $ChooseRmVertex$, which returns a vertex from the candidate vertex set C to remove. We propose a fast and effective heuristic for doing this task, which strikes a good balance between the time complexity and the quality of the selected vertex (w.r.t. the $loss$ value).

Local search algorithms usually need to select an element from a candidate set. Perhaps the most commonly used strategy is to choose the best element according to some criterion, which we refer to as "best-picking" heuristic. With a suitable criterion, this heuristic guides the search towards the most promising area, and is thus widely adopted in local search algorithms. Recent examples of such heuristics for MinVC include the max-gain pair selection heuristic in COVER [Richter *et al.*, 2007] and the minimum loss removing heuristic in NuMVC [Cai *et al.*, 2013]. More examples can be found in local search algorithms for other famous NP-hard problems, such as the Satisfiability problem [Selman *et al.*, 1992; Hoos and Stützle, 2004; Li and Huang, 2005; Cai and Su, 2013]. Indeed, a lot of works on local search have been focused on the criterion for filtering the candidate set and the function for comparing elements, and once this is done, they simply pick the best one. The "best-picking" heuristic works well in most cases, but not for massive data sets where the candidate set is usually very large and finding the best element is very time-consuming.

We propose a cost-effective heuristic called Best from Multiple Selections (BMS), for picking a good element from a set, and we use it for the $ChooseRmVertex$ function. For a set S, the BMS heuristic works as follows:

Choose k elements randomly with replacement from the set S, and then return the best one (w.r.t. some comparison function f), where k is a parameter.

A more formal description of the BMS heuristic is given in Algorithm 3. Let us look at how well the BMS heuristic approximates the "best-picking" heuristic. For a real number $\rho \in (0, 1)$, the probability of the event $E = \{$the f value of the element chosen by BMS is not greater than $\rho|S|$ elements

Algorithm 3: Best from Multiple Selection (BMS) Heuristic

```
Input: A set $S$, a parameter $k$, a comparison function $f$
//assume $f$ is a function such that we say an element is better
than another one if it has smaller $f$ value
Output: an element of $S$
1 $best$ :=a random element from $S$;
2 for $iteration := 1$ to $k - 1$ do
3     $r$ :=a random element from $S$;
4     if $f(r) < f(best)$ then $best := r$;
5 return $best$;
```

in the set $S\}$ is $Pr(E) \geq 1-(\frac{\rho|S|-1}{|S|})^k > 1-\rho^k$ (the first "$\geq$" because there might be the case that more than one elements in those $\rho|S|$ elements have the same f value, which is the minimum among f values of all the $\rho|S|$ elements).

For $ChooseRmVertex$, the comparison function f is simply the $loss$ function on vertices, and we set $k = 50$. Then, the probability that the BMS heuristic chooses a vertex whose $loss$ value is not greater than 90% vertices in C is $Pr(E) > 1 - 0.9^{50} > 0.9948$. The above calculations illustrate that the BMS heuristic returns a vertex of very good quality with a very high probability.

The complexity of the BMS heuristic is $O(k) = O(1)$, since k is a constant parameter. This is lower than $O(|C|)$ for the minimum loss heuristic used by previous local search algorithms for MinVC. Note that BMS is a generic heuristic and can be also applied to improve the time efficiency of local search algorithms for large scale instances of other problems.

5 Experimental Evaluation

In this section, we carry out extensive experiments to evaluate FastVC on a broad range of real world graphs, compared against the state of the art local search MinVC algorithm NuMVC.

5.1 Benchmarks

For our experiments, we collected all undirected simple graphs (not including DIMACS and BHOSLIB graphs) we could find from the Network Data Repository online [Rossi and Ahmed, 2015].[3] Many of these real world graphs have millions of vertices and dozens of millions of edges. Some of these benchmarks have recently been used in testing parallel algorithms for Maximum Clique and Coloring problems [Rossi and Ahmed, 2014; Rossi *et al.*, 2014].

The graphs in our experiments can be grouped into 10 classes, including biological networks, collaboration networks, facebook networks, interaction networks, infrastructure networks, amazon recommend networks, scientific computation networks, social networks, technological networks, and web lint networks, in the order of their appearance in the tables. There is also a group of temporal reachability networks, where the graphs are small (usually with several hundreds or several thousands of vertices) and the two algorithms find the same quality solution on all the graphs, and thus is not included in our experiment.

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

5.2 Experiment Setup

For comparisons, we use the NuMVC algorithm [Cai *et al.*, 2013] to represent the state of the art in solving MinVC (and also MaxIS) problem. Based on experiments on DIMACS and BHOSLIB benchmarks, NuMVC succeeds more reliably to find the optimal or best known solution at speeds at least several times faster than earlier MinVC and MaxIS algorithms [Cai *et al.*, 2013]. It is acknowledged as the latest breakthrough for MinVC solving in the literature [Fang *et al.*, 2014; Rosin, 2014; Jin and Hao, 2015]. Slightly better results have been reported for two algorithms that build on the top of NuMVC [Fang *et al.*, 2014; Cai *et al.*, 2015], but this does not materially change our conclusions below.

We implement our algorithm FastVC in the C++ programming language. NuMVC is open-source and also implemented in C++.[4] Both algorithms are complied by g++ (version 4.4.5) with the '-O3' option. For NuMVC, we adopt the parameter setting reported in [Cai *et al.*, 2013]. For the BMS heuristic in the $ChooseRmVertex$ function of FastVC, we set the k parameter to 50, as mentioned in the previous section. The experiments are carried out on a workstation under Ubuntu Linux, using 2 cores of i7-4800MQ 2.7 GHz CPU and 8 GByte RAM.

Both algorithm are executed 10 times on each instance with a time limit of 1000 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 comparison clearer, we also report the difference ("Δ") between the minimum size of vertex cover found by NuMVC and that found by FastVC. A positive Δ means FastVC finds a smaller vertex cover, and a negative Δ means NuMVC finds a smaller vertex cover. For some instances, NuMVC fails to find a vertex cover within the time limit, then the column for NuMVC is marked as "n/a" and the "Δ" column is marked as "max" (since FastVC finds one).

5.3 Experiment Results

We present the main experiment results in Tables 1 and 2. For the sake of space, we do not report the results on graphs with less than 1000 vertices, for which both algorithms find the same quality solutions quickly. Also, three extremely large graphs are not reported as they are out of memory for both algorithms.

From the results in Tables 1 and 2, we observe that:

1) Out of the 86 graphs, FastVC finds better vertex covers than NuMVC for 46 graphs, and finds the same quality solutions for 37 graphs. FastVC finds worse solutions only for 3 graphs.

2) The two algorithms have similar performance on two classes of benchmarks, namely the biological networks and interaction networks; for other classes of benchmarks, FastVC significantly outperforms NuMVC.

3) There are 10 graphs for which NuMVC fails to provide a vertex cover within the time limit, while FastVC finds vertex covers for all the graphs. This indicates the effectiveness of our construction heuristic. In fact, further observations show

[4]http://lcs.ios.ac.cn/~caisw/Code/NuMVC-Code.zip

Table 1: Experimental results on biological networks, collaboration networks, facebook networks, interaction networks, infrastructure networks and recommend networks

Graph	|V|	|E|	NuMVC C_{min} (C_{avg})	FastVC C_{min} (C_{avg})	Δ
bio-dmela	7393	25569	2630 (2630)	2630 (2630)	0
bio-yeast	1458	1948	456 (456)	456 (456)	0
ca-AstroPh	17903	196972	11483 (11483)	11483 (11483)	0
ca-citeseer	227320	814134	129193 (129196)	129193 (129193)	0
ca-coauthors-dblp	540486	15245729	472251 (472258)	472179 (472179)	72
ca-CondMat	21363	91286	12480 (12480)	12480 (12480)	0
ca-CSphd	1882	1740	550 (550)	550 (550)	0
ca-dblp-2010	226413	716460	121971 (121974)	121969 (121969)	2
ca-dblp-2012	317080	1049866	164952 (164956)	164949 (164949)	3
ca-Erdos992	6100	7515	461 (461)	461 (461)	0
ca-GrQc	4158	13422	2208 (2208)	2208 (2208)	0
ca-HepPh	11204	117619	6555 (6555)	6555 (6555)	0
ca-hollywood-2009	1069126	56306653	n/a	864052 (864052)	max
ca-MathSciNet	332689	820644	139982 (139990)	139951 (139951)	31
socfb-A-anon	3097165	23667394	n/a	375231 (375233)	max
socfb-B-anon	2937612	20959854	n/a	303048 (303049)	max
socfb-Berkeley13	22900	852419	17216 (17218)	17210 (17212.7)	6
socfb-CMU	6621	249959	4986 (4986.1)	4986 (4986.5)	0
socfb-Duke14	9885	506437	7683 (7683.6)	7683 (7683)	0
socfb-Indiana	29732	1305757	23320 (23326.5)	23315 (23317.1)	5
socfb-MIT	6402	251230	4657 (4657)	4657 (4657)	0
socfb-OR	63392	816886	36558 (36560.6)	36548 (36549.2)	10
socfb-Penn94	41536	1362220	31179 (31183.3)	31162 (31164.8)	17
socfb-Stanford3	11586	568309	8518 (8518)	8517 (8517.9)	1
socfb-Texas84	36364	1590651	28174 (28180.5)	28167 (28171.1)	7
socfb-UCLA	20453	747604	15225 (15227.1)	15223 (15224.3)	2
socfb-UConn	17206	604867	13231 (13233)	13230 (13231.5)	1
socfb-UCSB37	14917	482215	11262 (11263)	11261 (11263)	1
socfb-UF	35111	1465654	27319 (27323.5)	27306 (27309)	13
socfb-UIllinois	30795	1264421	24096 (24106.4)	24091 (24092.2)	5
socfb-Wisconsin87	23831	835946	18388 (18390.6)	18383 (18385.1)	5
ia-email-EU	32430	54397	820 (820)	820 (820)	0
ia-email-univ	1133	5451	594 (594)	594 (594)	0
ia-enron-large	33696	180811	12781 (12781)	12781 (12781)	0
ia-fb-messages	1266	6451	578 (578)	578 (578)	0
ia-reality	6809	7680	81 (81)	81 (81)	0
ia-wiki-Talk	92117	360767	17288 (17288.2)	17288 (17288)	0
inf-power	4941	6594	2203 (2203)	2203 (2203)	0
inf-roadNet-CA	1957027	2760388	n/a	1001273 (100131.2)	max
inf-roadNet-PA	1087562	1541514	n/a	555220 (555243)	max
rec-amazon	91813	125704	47655 (47672.6)	47606 (47606)	49

that the construction heuristic in FastVC outputs a vertex cover typically in one second.

4) The Δ value is greater than 10 for 30 graphs. Such big improvements on the size of vertex cover are very exciting, when compared to those in the literature of MinVC algorithms.

For 33 graphs where the two algorithms consistently find vertex covers of the same size, we compare their averaged running time to locate such a solution. The results are reported in Table 3, which show that FastVC is much faster than NuMVC.

6 Summary and Future Work

In this work, we develop a local search algorithm for MinVC called FastVC, based on two new heuristics. The first heuristic is a construction procedure which has a complexity of $O(|E|)$, compared to $O(|V|^2)$ for the construction heuristic used in previous local search algorithms for MinVC. The second one is the Best from Multiple Selections (BMS) heuristic, which approximates the minimum loss heuristic

Table 2: Experimental results on scientific computation networks, social networks, technological networks and web lint networks

Graph	|V|	|E|	NuMVC C_{min} (C_{avg})	FastVC C_{min} (C_{avg})	Δ
sc-ldoor	952203	20770807	n/a	856755 (856757)	max
sc-msdoor	415863	9378650	381569 (381575)	381558 (381559)	11
sc-nasasrb	54870	1311227	51244 (51246.7)	51244 (51247.3)	0
sc-pkustk11	87804	2565054	83911 (83911)	83911 (83912.5)	0
sc-pkustk13	94893	3260967	89218 (89222)	89217 (89220.6)	1
sc-pwtk	217891	5653221	207749 (207756)	207716 (207720)	33
sc-shipsec1	140385	1707759	117477 (117537)	117318 (117338)	159
sc-shipsec5	179104	2200076	147288 (147324)	147137 (147174)	151
soc-BlogCatalog	88784	2093195	20752 (20752)	20752 (20752)	0
soc-brightkite	56739	212945	21192 (21193.5)	21190 (21190)	2
soc-buzznet	101163	2763066	30613 (30613.2)	30625 (30625)	-12
soc-delicious	536108	1365961	85522 (85585.6)	85586 (85596.4)	-64
soc-digg	770799	5907132	103303 (103319)	103244 (103245)	59
soc-douban	154908	327162	8685 (8685)	8685 (8685)	0
soc-epinions	26588	100120	9757 (9757)	9757 (9757)	0
soc-flickr	513969	3190452	153343 (153353)	153272 (153272)	71
soc-flixster	2523386	7918801	96319 (96320.7)	96317 (96317)	2
soc-FourSquare	639014	3214986	90125 (90134.1)	90108 (90109.2)	17
soc-gowalla	196591	950327	84313 (84322.6)	84222 (84222.3)	91
soc-lastfm	1191805	4519330	78692 (78695)	78688 (78688)	4
soc-livejournal	4033137	27933062	n/a	1869046 (1869051.1)	max
soc-LiveMocha	104103	2193083	43430 (43432.8)	43427 (43427)	3
soc-pokec	1632803	22301964	n/a	843422 (843435)	max
soc-slashdot	70068	358647	22373 (22377)	22373 (22373)	0
soc-twitter-follows	404719	713319	2323 (2323)	2323 (2323)	0
soc-youtube	495957	1936748	146456 (146468)	146376 (146376)	80
soc-youtube-snap	1134890	2987624	277015 (277025)	276945 (276945)	70
tech-as-caida2007	26475	53381	3683 (3683)	3683 (3683)	0
tech-as-skitter	1694616	11094209	n/a	527185 (527196)	max
tech-internet-as	40164	85123	5700 (5700)	5700 (5700)	0
tech-p2p-gnutella	62561	147878	15682 (15682)	15682 (15682)	0
tech-RL-caida	190914	607610	74759 (74776.5)	74930 (74938.9)	-171
tech-routers-rf	2113	6632	795 (795)	795 (795)	0
tech-WHOIS	7476	56943	2284 (2284)	2284 (2284)	0
web-arabic-2005	163598	1747269	114464 (114472)	114426 (114427)	38
web-BerkStan	12305	19500	5384 (5384)	5384 (5384)	0
web-edu	3031	6474	1451 (1451)	1451 (1451)	0
web-google	1299	2773	498 (498)	498 (498)	0
web-indochina-2004	11358	47606	7300 (7300)	7300 (7300)	0
web-it-2004	509338	7178413	414738 (414755)	414671 (414676)	67
web-sk-2005	121422	334419	58199 (58205.5)	58173 (58173)	26
web-spam	4767	37375	2297 (2297)	2297 (2297)	0
web-uk-2005	129632	11744049	127774 (127774)	127774 (127774)	0
web-webbase-2001	16062	25593	2652 (2652)	2651 (2651.2)	1
web-wikipedia2009	1864433	4507315	n/a	648317 (648322)	max

Table 3: Comparison of running time on instances where all vertex covers found by the two algorithms have the same size. The reported time is the averaged running time over 10 runs.

Graph	NuMVC time	FastVC time	Graph	NuMVC time	FastVC time
bio-dmela	0.828	<0.01	soc-BlogCatalog	448.804	1.9
bio-yeast	<0.01	<0.01	soc-douban	1.374	0.06
ca-AstroPh	5.028	0.024	soc-epinions	95.104	0.138
ca-citeseer	153	1.074	soc-slashdot	774.653	0.208
ca-CondMat	80.604	0.27	soc-twitter-follows	0.798	0.19
ca-CSphd	<0.01	<0.01	tech-as-caida2007	1.074	0.08
ca-Erdos992	<0.01	<0.01	tech-internet-as	9.668	0.015
ca-GrQc	0.156	<0.01	tech-p2p-gnutella	22.358	0.011
ca-HepPh	23.224	<0.01	tech-routers-rf	0.014	<0.01
socfb-MIT	6.69	41.133	tech-WHOIS	0.454	<0.01
ia-email-EU	0.2	<0.01	web-BerkStan	8.862	25.807
ia-email-univ	<0.01	<0.01	web-edu	0.374	<0.01
ia-enron-large	48.738	0.049	web-google	<0.01	<0.01
ia-fb-messages	<0.01	<0.01	web-indochina-2004	17.242	0.091
ia-reality	<0.01	<0.01	web-spam	304.682	8.674
ia-wiki-Talk	510.583	8.6	web-uk-2005	0.285	0.06
inf-power	0.79	0.21			

very well and lowers the complexity from $O(|V|)$ to $O(1)$.

Thanks to these two heuristics, the FastVC algorithm performs much better than the state of the art algorithm NuMVC on massive graphs. Experiments on massive real world graphs show that, FastVC finds smaller vertex covers than NuMVC on most graphs. Also, FastVC finds a "good" (or at least not too bad) vertex cover for all the graphs within reasonable time (1000 seconds), while NuMVC fails to find a vertex cover on a considerable portion of the graphs.

This work takes a first step towards local search for MinVC on massive graphs, and also provides key insights about balance between complexity and quality of heuristics for massive data problems. In the future, we would like to design more efficient heuristic algorithms for MinVC as well as other graph problems on massive graphs.

Acknowledgement

This work is supported by China National 973 Program 2014CB340301, National Natural Science Foundation of China 61370072, 61370156 and 61472369. We would like to thank the anonymous referees for their helpful comments.

References

[Andrade *et al.*, 2008] Diogo Viera Andrade, Mauricio G. C. Resende, and Renato Fonseca F. Werneck. Fast local search for the maximum independent set problem. In *Workshop on Experimental Algorithms*, pages 220–234, 2008.

[Barabási and Albert, 1999] A.L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509, 1999.

[Cai and Su, 2013] Shaowei Cai and Kaile Su. Local search for Boolean Satisfiability with configuration checking and subscore. *Artif. Intell.*, 204:75–98, 2013.

[Cai *et al.*, 2010] Shaowei Cai, Kaile Su, and Qingliang Chen. EWLS: A new local search for minimum vertex cover. In *Proc. of AAAI-10*, pages 45–50, 2010.

[Cai *et al.*, 2011] Shaowei Cai, Kaile Su, and Abdul Sattar. Local search with edge weighting and configuration checking heuristics for minimum vertex cover. *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 *et al.*, 2015] Shaowei Cai, Jinkun Lin, and Kaile Su. Two weighting local search for minimum vertex cover. In *Proc. of AAAI-15*, pages 1107–1113, 2015.

[Dinur and Safra, 2005] I. Dinur and S. Safra. On the hardness of approximating minimum vertex cover. *Annals of Mathematics*, 162(2):439–486, 2005.

[Eubank *et al.*, 2004] Stephen Eubank, V. S. Anil Kumar, Madhav V. Marathe, Aravind Srinivasan, and Nan Wang. Structural and algorithmic aspects of massive social networks. In *Proc. of SODA-04*, pages 718–727, 2004.

[Fang *et al.*, 2014] Zhiwen Fang, Yang Chu, Kan Qiao, Xu Feng, and Ke Xu. Combining edge weight and vertex weight for minimum vertex cover problem. In *Proc. of FAW-14*, pages 71–81, 2014.

[Garey and Johnson, 1979] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. Freeman, San Francisco, CA, USA, 1979.

[Hoos and Stützle, 2004] H.H. Hoos and T. Stützle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann, San Francisco, CA, USA, 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.

[Karakostas, 2005] G. Karakostas. A better approximation ratio for the vertex cover problem. In *Proc. of ICALP-05*, pages 1043–1050, 2005.

[Kavalci *et al.*, 2014] Vedat Kavalci, Aybars Ural, and Dagdeviren. Distributed vertex cover algorithms for wireless sensor networks. *International Journal of Computer Networks & Communications (IJCNC)*, 6:95–110, 2014.

[Li and Huang, 2005] Chu Min Li and Wen Qi Huang. Diversification and determinism in local search for satisfiability. In *Proc. of SAT-05*, pages 158–172, 2005.

[Lu and Chung, 2006] L. Lu and F. Chung. *Complex Graphs and Networks*. American Math. Society, New York, USA, 2006.

[Papadimitrious and Steiglitz, 1982] C. H. Papadimitrious and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall, New York, USA, 1982.

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

[Richter *et al.*, 2007] Silvia Richter, Malte Helmert, and Charles Gretton. A stochastic local search approach to vertex cover. In *Proc. of KI-07*, pages 412–426, 2007.

[Rosin, 2014] Christopher D. Rosin. Unweighted stochastic local search can be effective for random CSP benchmarks. *CoRR*, abs/1411.7480, 2014.

[Rossi and Ahmed, 2014] Ryan A Rossi and Nesreen K Ahmed. Coloring large complex networks. *Social Network Analysis and Mining (SNAM)*, pages 1–52, 2014.

[Rossi and Ahmed, 2015] Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *Proc. of AAAI-15*, 2015.

[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 *WWW (Companion Volume)*, pages 365–366, 2014.

[Selman *et al.*, 1992] Bart Selman, Hector J. Levesque, and David G. Mitchell. A new method for solving hard satisfiability problems. In *Proc. of AAAI-92*, pages 440–446, 1992.

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