Article

Not peer-reviewed version

TIVC: An Efficient Local Search Algorithm for Minimum Vertex Cover in Large Graphs

Yu Zhang , Shengzhi Wang , Chanjuan Liu , Enqiang Zhu *

Posted Date: 31 July 2023

doi: 10.20944/preprints202307.2022.v1

Keywords: minimum vertex cover (MVC); local search; wireless sensor networks (WSNs); combinatorial optimization; large graphs

 doi:10.20944/preprints202307.2022.v1

Article

TIVC: An Efficient Local Search Algorithm for Minimum Vertex Cover in Large Graphs

Yu Zhang [1], Shengzhi Wang [2], Chanjuan Liu [3] and Enqiang Zhu [2,*]

1 Cyberspace Institute of Advanced Technology, Guangzhou University, Guangzhou 510006, China; zhangyu@e.gzhu.edu.cn
2 Institute of Computing Science and Technology, Guangzhou University, Guangzhou 510006, China; wallace_sz@163.com
3 School of Computer Science and Technology, Dalian University of Technology, Dalian 116024, China; chanjuan.cs.ai@gmail.com
* Correspondence: zhuenqiang@gzhu.edu.cn

Abstract: The minimum vertex cover (MVC) problem is a canonical **NP**-hard combinatorial optimization problem, aiming to find a smallest set of vertices such that every edge has at least one endpoint in the set, which has extensive applications in cyber security, scheduling, and monitoring link failures in wireless sensor networks (WSNs). Numerous local search algorithms have been proposed to obtain a "good" vertex cover. However, due to the **NP**-hard nature, it is challenging to efficiently solve the MVC problem, especially on large graphs. In this paper, we propose an efficient local search algorithm for MVC called TIVC, which is based on two main ideas: A 3-improvements (TI) framework with tiny perturbation, and an edge selection strategy. We conducted experiments on real-world large instances of a massive graph benchmark. Compared with two state-of-the-art MVC algorithms, TIVC shows superior performance in accuracy and possesses a remarkable ability to identify significantly smaller vertex covers on many graphs.

Keywords: minimum vertex cover (MVC); local search; wireless sensor networks (WSNs); combinatorial optimization; large graphs

1. Introduction

Given an undirected graph $G = (V, E)$, a *vertex cover* (VC) $C \subseteq V$ of G is a subset of vertices such that every edge $e \in E$ has at least one endpoint belonging to C. The *minimum vertex cover* (MVC) problem is to find a VC with the smallest size in a graph, which is a classical **NP**-hard problem with an approximation factor of 1.3606 [1]. MVC plays an important role in graph theory for its extensive applications, including scheduling [2], cyber security [3], and wireless sensor networks (WSNs) [4]. For example, a WSN can be modeled as an undirected graph, where vertices and edges represent infrastructures and communication links, respectively. Then, elements in VCs can be used for various purposes such as monitoring link failures, facility location, clustering, and data aggregation, since each communication link (edge) is incident with at least one vertex in a VC. Figure 1 shows an example of simulating a wireless sensor network using a unit disk graph[5], where each vertex is the center of a circle and there is an edge between it and the other vertices within its radius.

The MVC problem has been extensively studied and many algorithms have been proposed, including exact and approximate algorithms. Regarding exact algorithms, branch-and-reduce methods currently have the best time complexity [6,7]. However, the exact algorithms are still exponential-time, which cannot solve the MVC problem in a reasonable time, especially on large graphs. Therefore, approximate algorithms are proposed to solve MVC. Greedy algorithms are the common method used for approximately solving intractable problems, such as connected dominating sets [8], weighted vertex covers [9], and independent sets [10]. While greedy algorithms can quickly produce feasible solutions, the quality of solutions is generally not high enough to meet real-world requirements.

In practice, tackling intractable problems often resorts to heuristic approaches for obtaining a high-quality solution within a reasonable time, and a number of such algorithms have been proposed

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 31 July 2023 doi:10.20944/preprints202307.2022.v1

to address various problems, such as job shop scheduling [11], partition coloring [12], and critical nodes problem [13]. Local search is one of the extensively studied heuristics for solving **NP**-hard problems [14–18]. Regarding the MVC problem, it has been shown that the local search outperforms other heuristics [19]. The primary idea of local search algorithms for solving graph theory problems can be described as follows: initiate with a feasible solution and iteratively update it by removing, adding, or swapping vertices until a cutoff time is reached. A common strategy is (j, k)-swaps, i.e. removing j vertices from a solution and adding k vertices to it. We refer to a (j, k)-swap as a j-improvement [20]. The local search algorithms have the advantages of simple implementation and effective performance. However, they do suffer from a few challenges: The cycling phenomenon (i.e., revisiting recently visited vertices) [21,22] leads to the algorithm wasting too much computational time, resulting in a local optimum; moreover, complex vertex selection strategies may diminish the efficiency of the local search, resulting in a poor performance in large graphs. To address these issues, researchers have proposed many strategies, which will be described in detail in Section 3.

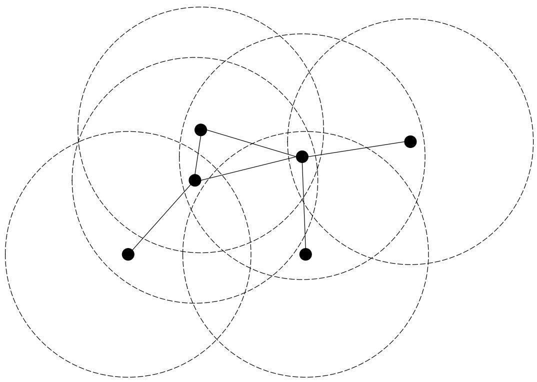

Figure 1. An example of a WSN.

This paper proposes an efficient algorithm for the MVC problem on large graphs named TIVC, which has two main ideas. The first one is a 3-improvements framework with a tiny perturbation. State-of-the-art local search algorithms for MVC are based on 2-improvements frameworks, and both 2-improvements and 3-improvements aim to search for a feasible solution of size $(k-1)$ based on a k-sized feasible solution. However, 3-improvements have a chance to directly search for a solution of size $(k-2)$ after adding the first vertex, which enhances the efficiency of the algorithm. Moreover, during the vertex-removing phase, the third vertex is removed from the solution in a perturbation manner, which expands the algorithm's search space and enhances its ability to deal with local optima dilemmas. Second, we use an effective edge selection strategy to accelerate search speed. When selecting uncovered edges for covering, it is essential to consider both the effectiveness and complexity of the selection strategy. During the vertex-removing phase, different vertex selection strategies can result in different numbers of uncovered edges (for example, a greedy-based vertex selection strategy generates fewer uncovered edges, while a random vertex selection strategy produces more). Adding vertices to the solution by traversing all uncovered edges is not only time-consuming but can also easily lead the algorithm into local optima. Based on these considerations, we combine the *edge age based best from multiple selections* (EABMS) technique [19] with a random vertex selection method to choose the uncovered edges to be covered.

We conduct experiments to compare TIVC with state-of-the-art local search algorithms for MVC on the Network Repository benchmark, including 72 real-world large instances. TIVC shows the best accuracy performance and significantly outperforms other algorithms on many instances.

The remainder of this paper is organized as follows. Section 2 presents basic definitions. Section 3 gives a brief review of the related work of MVC. Section 4 describe the TIVC algorithm. Section 5 is devoted to the design and analysis of experiments, and Second 6 provides concluding remarks.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 31 July 2023 doi:10.20944/preprints202307.2022.v1

2. Preliminaries

This section introduces some preliminary knowledge.

2.1. Notations and Terminologies

Denote by $G = (V, E)$ an undirected graph with *vertex set* V and *edge set* E. For an edge $e = (u, v)$, the two vertices u and v are called *endpoints* of e. A vertex is *adjacent* to another vertex if they are the two endpoints of an edge, and one is called a *neighbor* of the other. An edge is *incident* with each of its endpoints. The set consisting of all neighbors of a vertex $v \in V$, denoted by $N(v)$, is the *neighborhood* of v, and $N[v] = N(v) \cup \{v\}$ is the *closed neighborhood* of v. The degree of v is the number of edges incident with v. For a vertex set $S \subseteq V$, let $N[S] = \bigcup_{v \in S} N[v]$ and $N(S) = N[S] \setminus S$.

For a graph $G = (V, E)$ and a set of vertices $S \subseteq V$, an edge $e \in E$ is *covered* by S if at least one endpoint of e belongs to S; otherwise, e is *uncovered* by S. If all edges of G are covered by S, then S is called a *vertex cover* (VC) of G. A VC with the smallest cardinality is called a *minimum vertex cover* (MVC) of G. Note that a graph $G = (V, E)$ may have more than one MVC. We use $E_u(S) \subseteq E$ to denote the set of edges uncovered by S, and use $E_c(S) \subseteq E$ to denote the set of edges covered by S. The MVC problem is to find a MVC from a graph.

2.2. Local Search

From this section, C represents a candidate (or partial) solution of the MVC problem. The general scheme of local search for MVC is to construct an initial VC first and then iteratively improve the solution to a smaller one by vertex swapping. Generally, local search algorithms use *gain(v)* and *loss(v)* to measure the importance of a vertex v, where *gain(v)* denotes the number of edges uncovered by C but covered by $C \cup \{v\}$, and *loss(v)* the number of edges covered by C but uncovered by $C \setminus \{v\}$. The *age* of a vertex v, denoted by *age(v)*, is the number of the steps since it was last removed from C. *age* values are usually used to break ties, where ties mean the existence of multiple vertices with the same *gain* or *loss*. In addition, the age of an edge e, denoted by *age(e)*, is the number of steps since it was last uncovered by C, which is often used as a criterion for selecting edges [19].

3. Related Work

This section provides a brief review on heuristic algorithms for MVC. In 2013, Cat et al. [23] proposed a two-stage strategy that allows to select a pair of vertices separately and exchange vertices in two stages, based on which a NuMVC algorithm for MVC is developed, addressing the drawback (time-consuming) of previous algorithms that requires selecting vertices simultaneously [21,24,25]. However, with the rapid development of the Internet and the widespread deployment of sensors, the size of datasets has dramatically increased, and many algorithms fail to solve MVC on large instances. For this, Cai et al. [26] introduced the *Best from Multiple Selections* (BMS) heuristic, which randomly samples k vertices in C and removes one with the minimum *loss* value from C. This heuristic aims to obtain a trade-off between efficiency and accuracy. Based on BMS, an algorithm named FastVC is developed for solving MVC well on large instances. By combining BMS and the best-picking strategy [23], Ma et al. [27] proposed a best-picking with a noisy strategy and developed an algorithm NoiseVC; they also proposed a *BMS with random walk strategy* (WalkBMS) in another literature [28], to handle the issue that FastVC easily gets trapped in a local optimum. Subsequently, Cai et al. [29] proposed an improved version of FatVC, named FastVC2+p, by integrating some processing techniques and initial solution construction methods. In 2021, Quan et al. [19] proposed a new edge weighting method based on edge age (EABMS), which randomly samples a edges in $E_u(C)$ and selects one edge with the maximum *age* value for covering (by adding one of its endpoints to C). Based on EABMS, an algorithm EAVC and its variant EAVC2+p are developed for MVC. Both EAVC and EAVC2+p showed superiority on large graphs compared with FastVC and its variants. To date, FastVC, EAVC, and their variants are

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 31 July 2023 doi:10.20944/preprints202307.2022.v1

state-of-the-art MVC local search algorithms for large instances. To demonstrate the effectiveness of TIVC, we compared our algorithm with the baseline algorithms, i.e., FastVC and EAVC.

4. Main Algorithm

In this section, we describe our algorithm TIVC. We first introduce the top-level architecture of TIVC, and then describe the algorithm in detail.

4.1. Top-Level Architecture

The top-level architecture of TIVC is shown in Algorithm 1. TIVC starts with constructing an initial VC C for the graph G (line 1), and then enters a loop for finding a VC as small as possible within a given cutoff time (lines 2–10). Specifically, when obtaining a VC C, it updates the best solution C^* and then removes a vertex from C (lines 3–6). If C is not a feasible solution, then the algorithm iteratively exchanges vertices until C becomes a VC. First, it removes vertices from C until $|C| = |C^*| - 3$ (lines 7). Next, it selects an uncovered edge and adds one of its endpoints to C (line 8). If C remains infeasible, it selects another vertex adding to C in the same way (lines 9–10). Finally, the best-found vertex cover C^* is returned when the cutoff time is reached (line 11).

Algorithm 1: TOP-Level of TIVC

Input: A graph $G = (V, E)$, the *cutoff* time
Output: A vertex cover C^* of G
1 $C \leftarrow ConstructVC(G)$;
2 **while** *elapsed_time* < *cutoff* **do**
3   **if** C *covers all edges* **then**
4         $C^* \leftarrow C$;
5   remove a vertex from C;
6   continue;
      end
7   remove vertices from C until $|C| = |C^*| - 3$;
8   choose an uncovered edge and adds one of its endpoints to C;
9   **if** $\exists$ *uncovered edges* **then**
10   choose an uncovered edge and adds one of its endpoints to C;
      end
  end
11 **return** C^*;

4.2. The TIVC algorithm

Our TIVC algorithm is shown in Algorithm 2, which encompasses two stages, i.e. construction and search.

In the construction stage, the algorithm constructs an initial VC C of G (line 1) by EdgeGreedyVC [19,26], which is a commonly used approach for MVC algorithms. The process starts with an empty set C and proceeds iteratively by checking and covering edges to extend C. Once a VC is obtained, redundant vertices are removed from C, where redundant vertices are those with *loss=0* and removing them does not produce new uncovered edges.

In the search stage, the algorithm attempts to search for a VC smaller than the current C^*. At the beginning of the algorithm, it repeatedly removes a vertex with the minimum *loss* from C, until C is not a VC (lines 3–6). Second, the algorithm repeatedly performs vertex swapping. Each swapping step contains a removing phase (lines 7–11) and an adding phase (lines 12–17). In the removing phase, the first vertex u_1 is selected by the BMS heuristic and removed from C (lines 7–8); then the second vertex is selected randomly in C and removed from C to perturb the solution slightly (lines 9–10). The above implementation lead to $|C^*| = |C| - 3$. In the adding phase, it first selects an uncovered

Algorithm 2: TIVC

Input: A graph $G = (V, E)$, the *cutoff* time
Output: A vertex cover C^* of G
1 $C \leftarrow ConstructVC(G)$;
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, breaking ties by *age*;
6   continue;
    end
7     $u_1 \leftarrow$ a vertex in C selected by BMS heuristic, breaking ties by *age*;
8     $C \leftarrow C \setminus \{u_1\}$;
9     $u_2 \leftarrow$ a random vertex in C; `// tiny perturbation`
10     $C \leftarrow C \setminus \{u_2\}$;
11     $e \leftarrow$ a uncovered edge in $E_u(C)$ selected by BMS heuristic;
12     $w_1 \leftarrow$the endpoint of e with greater gain, breaking ties by *age*;
13     $C \leftarrow C \cup \{w_1\}$;
14   **if** $E_u(C) \neq \emptyset$ **then**
15         $e \leftarrow$ a random uncovered edge in $E_u(C)$;
16         $w_2 \leftarrow$the endpoint of e with greater gain, breaking ties by *age*;
17         $C \leftarrow C \cup \{w_2\}$;
    end
end
18 **return** C^*;

edge $e \in E_u(C)$ and adds the vertex with greater *gain* in its endpoints to C (lines 11–13); if there are uncovered edges ($E_u(C) \neq \emptyset$), then it choose one edge in $E_u(C)$ randomly and add the vertex with greater *gain* to C (lines 14–17). Finally, the best-found VC C^* is returned when the cutoff time is reached (line 18).

4.3. Complexity Analysis

In this section, we analyze the time complexity of TIVC. For a given graph $G = (V, E)$, let $|V| = n, |E| = m$.

TIVC (Algorithm 2) runs in $O(m + n)$. First, the *ConstructVC* procedure (line 1) constructs an initial solution by EdgeGreedyVC, which has a time complexity of $O(m)$ [26]. Second, lines 3–6 take $O(m + n)$ time since the procedure traverses E and V once. Third, lines 7–17 take $O(m)$ time because the time complexity of BMS and EABMS has already been proven to be $O(1)$ [19,26]; the time complexity of removing a vertex from a solution C and adding a vertex to C are $O(1)$, while line 14 need to traverse E once for checking the condition whether $E_u(C) \neq \emptyset$. Thus, the time complexity of TIVC is $O(m + n)$. $\square$

5. Results and Discussion

In this section, we evaluate TIVC on the Network Repository benchmark[1] [30]. This benchmark includes enormous amounts of graphs from various areas. To assess the performance of TIVC on large graphs, we specifically selected instances with vertex numbers ranging from 10^4 to 10^7, encompassing 72 instances.

[1] https://networkrepository.com

 doi:10.20944/preprints202307.2022.v1

5.1. Experiment Setup

TIVC is implemented in C++ and compiled by gcc 7.1.0 with the '-O3' optimization option. All experiments are run under CentOS Linux release 7.6.1810 with Intel(R) Xeon(R) Gold 6254 CPU@3.10GHz with 128GB RAM. The parameters of FastVC and EAVC are set to the same as those used in the original literature [19,26]. TIVC incorporates two tunable parameters: k for the BMS and a for the EABMS. These parameters are set to 50 and 24, respectively, aligning with the settings of EAVC.

We compare TIVC with two state-of-the-art local search algorithms, FastVC [26] and EAVC [19], for MVC. Both the two algorithms are designed to solve large instances for MVC. FastVC combines the two-stage exchange framework and the BMS heuristic to balance the algorithm's accuracy and efficiency, which achieves well performance on large graphs. EAVC is based on the two-stage framework, and combines WalkBMS and EABMS to provide a good guidance for improving the quality of solutions (and also increasing the diversity of solutions) in the vertex and edge selection phase.

Table 1 shows the details of these three algorithms. The construction procedures of these three algorithms are based on EdgeGreedyVC. For vertex selection, FastVC, EAVC, and TIVC utilize BMS, WalkBMS, and BMS+Random strategies, respectively. Regarding edge selection, FastVC, EAVC, and TIVC use Random, EABMS, and EABMS+Random strategies, respectively. In addition, FastVC and EAVC are both based on 2-improvements, while TIVC is based on 3-improvements. The codes of FastVC[2] and EAVC[3] were open online.

Table 1. Equipment of three algorithms.

Algorithms	FastVC	EAVC	TIVC
Construction	EdgeGreedyVC	EdgeGreedyVC	EdgeGreedyVC
Vertex Selection	BMS	WalkBMS	BMS+Random
Edge Selection	Random	EABMS	EABMS+Random
k-improvement	2	2	3

For each instance, all algorithms are executed 10 times with seeds $1, 2, 3, \ldots, 10$. The cutoff time for each run is set at 1,000 seconds. For each instance, we present the best (i.e., smallest) solution as Min, the average solution as Avg, and the average running time (over the 10 runs) as t_{avg}. Furthermore, we report the difference between the best solution found by TIVC and the solutions obtained by other algorithms as Δ.

5.2. Experimental Result

Results on the Network Repository benchmark are reported in Table 2. TIVC shows superior performance in terms of accuracy on almost all instances, outperforming both the FastVC and EAVC algorithms. Specifically, TIVC obtains the best solution on 60 (out of 78) instances, while FastVC and EAVC obtain 28 and 39 best solutions, respectively. In particular, TIVC shows a remarkable ability to find strictly optimal solutions, in total 28 such solutions. In comparison, FastVC and EAVC can only find 5 and 7 strictly optimal solutions, respectively. Regarding average solution, TIVC also outperforms the other algorithms. FastVC, EAVC, and TIVC obtain the optimal average VC on 11, 15, and 26 instances, respectively. In addition, for large instances with 10^7 vertices, TIVC performs remarkably well and finds much smaller VCs than other algorithms on many instances.

---

[2] http://lcs.ios.ac.cn/~caisw/VC.html
[3] https://github.com/quancs/EAVC

Table 2. The results on the Network Repository benchmark.

Instance	Graph_Properties		FastVC				EAVC				TIVC		
	$\lvert V\rvert$	$\lvert E\rvert$	*Min*	*Avg*	*t_avg*	Δ	*Min*	*Avg*	*t_avg*	Δ	*Min*	*Avg*	*t_avg*
cit-HepTh	22,908	2,444,798	**18155**	18155	0.054	0	**18155**	18155	0.123	0	**18155**	18155.4	0.12
as-22july06	22,963	48,436	**3303**	3303	0.005	0	**3303**	3303	2.41	0	**3303**	3303.6	0.005
cit-HepPh	28,093	3,148,447	**22589**	22589	0.344	0	**22589**	22589	0.214	0	**22589**	22589.2	0.371
cond-mat-2005	39,577	175,693	**23106**	23106	0.087	0	**23106**	23106	0.047	0	**23106**	23106.4	0.065
soc-Epinions1	75,879	405,740	**22280**	22280	0.136	0	**22280**	22280	0.105	0	**22280**	22280.5	0.099
soc-Slashdot0811	77,360	469,180	**24046**	24046	0.309	0	**24046**	24046	0.15	0	**24046**	24046.4	0.141
soc-Slashdot0902	82,168	504,230	**25770**	25770.2	0.285	0	**25770**	25772.3	308.62	0	**25770**	25770.6	265.889
luxembourg_osm	114,599	119,666	**56936**	56937.6	186.235	0	**56936**	56937.9	37.136	0	**56936**	56936.5	68.115
wave	156,317	1,059,331	119306	119445.9	659.295	-16	119302	119382.1	561.156	-12	**119290**	119331.3	607.951
rec-dating	168,791	17,351,416	89839	89853.3	302.801	-47	89806	89811.4	752.091	-14	**89792**	89799.7	798.948
caidaRouterLevel	192,244	609,066	75433	75443.8	6.168	-260	75199	75215.8	235.861	-26	**75173**	75190.5	933.96
rec-libimseti-dir	220,970	17,233,144	94198	96275.5	479.417	-502	93704	93708	560.666	-8	**93696**	93701.4	742.141
coAuthorsCiteseer	227,320	814,134	**129193**	129193	1.321	0	**129193**	129193	0.579	0	**129193**	129193.6	0.597
amazon0302	262,111	899,792	168554	168557.9	574.718	-9	168547	168552.5	467.84	-2	**168545**	168550.6	556.788
email-EuAll	265,009	364,481	18317	18317	0.062	-1	**18316**	18316.1	0.067	0	**18316**	18317.2	0.067
Ga41As41H72	268,096	9,110,190	237101	237985	997.207	-3593	233550	233604.9	636.463	-42	**233508**	233632.2	397.482
citationCiteseer	268,495	1,156,647	118180	118184.6	12.267	-45	**118135**	118147.6	84.145	0	**118135**	118146	440.379
web-Stanford	281,903	1,992,636	118879	118895.1	637.112	-276	118613	118625.2	444.525	-10	**118603**	118616.2	878.722
coAuthorsDBLP	299,067	977,676	**155618**	155618	5.936	0	**155618**	155618	1.663	0	**155618**	155619	1.126
ca-dblp-2012	317,080	1,049,866	**164949**	164949	3.737	0	**164949**	164949	1.29	0	**164949**	164949.7	0.762
cnr-2000	325,557	2,738,969	96091	96104.1	368.162	-356	95778	95798.4	768.95	-43	**95735**	95769.6	930.457
web-NotreDame	325,729	1,090,108	74094	74104.3	128.563	-167	73943	73948.3	394.219	-16	**73927**	73931.7	895.304
amazon0312	400,727	2,349,869	261594	261598.7	840.595	-3	261596	261602.4	422.345	-5	**261591**	261598.5	834.251
amazon0601	403,394	2,443,408	266579	266586.3	461.456	-14	266567	266572.5	346.386	-2	**266565**	266572.2	662.399
amazon0505	410,236	2,439,437	267256	267260.3	682.728	-8	267252	267257.4	505.845	-4	**267248**	267254.7	715.894
coPapersCiteseer	434,102	16,036,720	**386106**	386106	9.861	0	**386106**	386106	5.758	0	**386106**	386106.9	6.148
ca-coaut*dblp	540,486	15,245,729	**472179**	472179	20.443	0	**472179**	472179	5.014	0	**472179**	472179.9	4.421
coPapersDBLP	540,486	15,245,729	**472179**	472179	35.569	0	**472179**	472179	8.36	0	**472179**	472179.9	8.034
web-BerkStan	685,230	6,649,470	278906	278934.2	799.023	-1706	277209	277228	546.692	-9	**277200**	277219.2	843.83
rec-epinion	755,200	13,396,042	100435	100444.3	5.654	-424	**100011**	100016.7	2.098	0	**100011**	100017.3	113.183
eu-2005	862,664	16,138,468	412377	412397.6	879.536	-1362	**411007**	411040.6	906.16	8	411015	411034.9	850.332
web-Google	875,713	4,322,051	346920	346924.7	131.099	-247	**346672**	346680.8	256.536	1	346673	346680.4	283.372
ldoor	952,203	22,785,136	899422	899423.2	292.146	-2	**899420**	899421	41.022	0	**899420**	899421.3	81.556
inf-roadNet-PA	1,087,562	1,541,514	**555231**	555251.8	662.74	27	555260	555316	379.001	-2	555258	555315.9	868.146
rt-retweet-crawl	1,112,702	2,278,852	81042	81044.6	79.529	-1	**81041**	81041.8	0.736	0	**81041**	81042	59.159
soc-youtube-snap	1,134,890	2,987,624	**276945**	276945	12.696	0	**276945**	276945.7	5.081	0	**276945**	276946.2	6.621
soc-lastfm	1,191,805	4,519,330	**78688**	78688	0.337	0	**78688**	78688	0.673	0	**78688**	78688.3	0.684
in-2004	1,382,867	13,591,473	487189	487237.8	902.545	-680	**486490**	486519	926.078	19	486509	486519	867.074
tech-as-skitter	1,694,616	11,094,209	527163	527201.2	410.913	-1671	525494	525515.5	520.156	-2	**525492**	525515.5	497.472
soc-flickr-und	1,715,255	15,555,041	**474637**	474637.5	233.62	0	**474637**	474637.9	94.72	0	**474637**	474638.5	69.201
inf-roadNet-CA	1,957,027	2,760,388	**1001317**	1001341	901.745	154	1001473	1001525	437.528	-2	1001471	1001513	749.759
web-baidu-baike	2,140,198	17,014,946	637106	637110.2	506.658	-93	637014	637019.8	330.014	-1	**637013**	637021	384.521
packing*b050	2,145,839	17,488,243	1624945	1625325	996.99	-1500	1624191	1625500	997.79	-746	**1623445**	1625416	997.741
tech-ip	2,250,498	21,643,497	**67007**	67007	1.349	0	**67007**	67007	4.918	0	**67007**	67007.4	2.907
soc-flixster	2,523,386	7,918,801	**96317**	96317	1.067	0	**96317**	96317	1.555	0	**96317**	96317.9	1.35
socfb-B-anon	2,937,612	20,959,854	**303048**	303048.9	42.414	0	**303048**	303048.2	3.171	0	**303048**	303048.7	3.212
soc-orkut	2,997,166	106,349,209	2171329	2171379	996.622	-116	2171270	2171301	997.663	-57	**2171213**	2171291	993.392
soc-orkut-dir	3,072,441	117,185,083	2233961	2234015	996.909	-103	**2233775**	2233820	997.141	83	2233858	2233929	996.253
socfb-A-anon	3,097,165	23,667,394	375231	375232.8	27.408	-1	**375230**	375230.9	100.17	0	**375230**	375230.9	75.813
patents	3,750,822	14,970,766	1673977	1674016	982.936	-377	**1673562**	1673615	969.758	38	1673600	1673632	976.264
soc-livejournal	4,033,137	27,933,062	1869043	1869052	928.872	-61	1868986	1868991	871.122	-4	**1868982**	1868991	860.125
delaunay_n22	4,194,304	12,582,869	2873973	2874015	999.086	-766	2873305	2873348	999.368	-98	**2873207**	2873239	999.194
ljournal-2008	5,363,186	49,514,271	2393023	2393035	948.562	-357	**2392664**	2392681	879.101	2	2392666	2392682	848.51
soc-ljournal-2008	5,363,186	49,514,271	2392992	2393038	950.129	-327	**2392660**	2392677	780.641	5	2392665	2392679	873.161
rel9	5,921,786	23,667,162	**273993**	273993.4	274.612	0	**273993**	273993.5	152.691	0	**273993**	273993.6	213.996
sc-rel9	5,921,786	23,667,162	**273993**	273993.3	333.193	0	**273993**	273993.4	258.317	0	**273993**	273993.6	198.414
soc-live*groups	7,489,073	112,305,407	1841367	1841386	907.272	-306	**1841061**	1841077	503.852	0	**1841061**	1841078	643.416
delaunay_n23	8,388,608	25,165,784	**5753835**	5754557	999.966	8986	5799719	5801179	999.987	-36898	5762821	5763597	999.988
friendster	8,658,744	45,671,471	1038252	1038257	588.53	-13	**1038239**	1038242	374.201	0	**1038239**	1038244	309.584
relat9	9,746,232	38,955,401	**274297**	274297	4.191	0	**274297**	274297	139.004	0	**274297**	274297.1	78.577
inf-germany_os	11,548,845	12,369,181	**5710522**	5710676	999.929	3934	5777786	5786140	999.989	-63330	5714456	5715014	999.982
hugetrace-00010	12,057,441	18,082,179	**6650729**	6754798	1000	111592	6914568	6924276	1000	-152247	6762321	6764085	1000
road_central	14,081,816	16,933,413	6902108	6911566	999.999	-11563	6944280	6945766	999.994	-53735	**6890545**	6895530	999.988
hugetrace-00020	16,002,413	23,998,813	9293370	9334922	1000	-53859	9321757	9333712	1000	-82246	**9239511**	9240936	999.998
delaunay_n24	16,777,216	50,331,601	11850819	11867478	1000	-27541	11871283	11874602	999.988	-48005	**11823278**	11824566	999.991
hugebu*00000	18,318,143	27,470,081	10469546	10498559	1000	-52038	10508631	10511251	1000	-91123	**10417508**	10432354	1000
uk-2002	18,483,186	261,787,258	6642980	6650452	999.997	-54848	6588420	6588632	999.546	-288	**6588132**	6588738	999.596
hugebu*00010	19,458,087	29,179,764	11667812	11695490	1000	-348369	11352764	11360173	1000	-33321	**11319443**	11328490	1000
hugebu*00020	21,198,119	45,671,471	12658142	12668880	1000	-300046	12406556	12407781	1000	-48460	**12358096**	12359528	1000
inf-road-usa	23,947,347	28,854,312	12022434	12027594	1000	-72203	11989552	11991769	999.986	-39321	**11950231**	11952066	999.992
inf-europe_os	50,912,018	54,054,660	25910226	25912219	1000	-13451	25918018	25924685	999.998	-21243	**25896775**	25898696	999.987
socfb-uci-uni	58,790,782	92,208,195	866768	866768	43.182	-2	**866766**	866766	26.199	0	**866766**	866767.4	24.497

Δ is equal to the TIVC's *Min* minus the corresponding algorithm's *Min*.

5.3. Discussion

The experimental results demonstrate the effectiveness of our TIVC algorithm for solving MVC on large graphs. We can observe that the *t_avg* values of all three algorithms are almost very close to 1,000 seconds on instances with 10^7 vertices, which indicates that the algorithms are less likely to fall into a local optimum on instances of this scale. For instances of this scale, 3-improvement exhibits a clear advantage. This is because the 3-improvement search framework can sometimes directly find a new solution of size $k-2$ based on a k-sized solution, whereas 2-improvement cannot achieve this.

 doi:10.20944/preprints202307.2022.v1

6. Conclusions

In this paper, we propose an efficient local search algorithm for the MVC problem called TIVC, which consists of a 3-improvements framework with tiny perturbation and an edge selection strategy. The experimental results show that TIVC significantly outperforms state-of-the-art algorithms for MVC on large graphs. In the future, we would like to study the ideas for solving other graph theory problems on large graphs.

Author Contributions: Conceptualization, Y.Z. and E.Z.; methodology, Y.Z.; software, Y.Z.; validation, Y.Z., E.Z. and S.W.; formal analysis, Y.Z.; investigation, Y.Z. and S.W.; resources, E.Z.; data curation, Y.Z.; writing—original draft preparation, Y.Z., S.W. and C.L.; writing—review and editing, E.Z. and C.L.; visualization, S.W.; supervision, E.Z.; project administration, E.Z.; funding acquisition, E.Z.

Funding: This work was supported in part by the National Natural Science Foundation of China under Grants(62272115),in part by the Natural Science Foundation of Guangdong Province of China under Grant 2021A 1515011940,in part by Science and Technology Projects in Guangzhou.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data used in this study are openly available at https://networkrepository.com, reference number [30].

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

WSN	Wireless sensor network
WSNs	Wireless sensor networks
VC	Vertex cover
TI	3-improvements
MVC	Minimum vertex cover
MIS	Maximum independent set
BMS	Best from Multiple Selections heuristic
WalkBMS	BMS with random walk strategy
EABMS	Edge age based best from multiple selections

References

1. Dinur, I.; Safra, S. On the hardness of approximating minimum vertex cover. *Annals of mathematics* **2005**, pp. 439–485.
2. Bansal, N.; Khot, S. Inapproximability of hypergraph vertex cover and applications to scheduling problems. International Colloquium on Automata, Languages, and Programming. Springer, 2010, pp. 250–261.
3. Javad-Kalbasi, M.; Dabiri, K.; Valaee, S.; Sheikholeslami, A. Digitally annealed solution for the vertex cover problem with application in cyber security. ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2019, pp. 2642–2646.
4. Kavalci, V.; Ural, A.; Dagdeviren, O. Distributed Vertex Cover Algorithms For Wireless Sensor Networks. *International Journal of Computer Networks & Communications* **2014**, *6*.
5. Clark, B.N.; Colbourn, C.J.; Johnson, D.S. Unit disk graphs. *Discrete Mathematics* **1990**, *86*, 165–177. doi:https://doi.org/10.1016/0012-365X(90)90358-O.
6. Akiba, T.; Iwata, Y. Branch-and-reduce exponential/FPT algorithms in practice: A case study of vertex cover. *Theoretical Computer Science* **2016**, *609*, 211–225.
7. Xiao, M.; Nagamochi, H. Exact algorithms for maximum independent set. *Information and Computation* **2017**, *255*, 126–146.
8. Li, Y.; Thai, M.T.; Wang, F.; Yi, C.W.; Wan, P.J.; Du, D.Z. On greedy construction of connected dominating sets in wireless networks. *Wireless Communications and Mobile Computing* **2005**, *5*, 927–932.

9. Bouamama, S.; Blum, C.; Boukerram, A. A population-based iterated greedy algorithm for the minimum weight vertex cover problem. *Applied Soft Computing* **2012**, *12*, 1632–1639.
10. Feo, T.A.; Resende, M.G.; Smith, S.H. A greedy randomized adaptive search procedure for maximum independent set. *Operations Research* **1994**, *42*, 860–878.
11. Viana, M.S.; Morandin Junior, O.; Contreras, R.C. A Modified Genetic Algorithm with Local Search Strategies and Multi-Crossover Operator for Job Shop Scheduling Problem. *Sensors* **2020**, *20*.
12. Zhu, E.; Jiang, F.; Liu, C.; Xu, J. Partition independent set and reduction-based approach for partition coloring problem. *IEEE Transactions on Cybernetics* **2020**, *52*, 4960–4969.
13. Liu, C.; Ge, S.; Zhang, Y. Identifying the cardinality-constrained critical nodes with a hybrid evolutionary algorithm. *Information Sciences* **2023**, *642*, 119140. doi:https://doi.org/10.1016/j.ins.2023.119140.
14. Dahlum, J.; Lamm, S.; Sanders, P.; Schulz, C.; Strash, D.; Werneck, R.F. Accelerating local search for the maximum independent set problem. Experimental Algorithms: 15th International Symposium, SEA 2016, St. Petersburg, Russia, June 5-8, 2016, Proceedings 15. Springer, 2016, pp. 118–133.
15. KhudaBukhsh, A.R.; Xu, L.; Hoos, H.H.; Leyton-Brown, K. SATenstein: Automatically building local search SAT solvers from components. *Artificial Intelligence* **2016**, *232*, 20–42.
16. Song, T.; Liu, S.; Tang, X.; Peng, X.; Chen, M. An iterated local search algorithm for the University Course Timetabling Problem. *Applied Soft Computing* **2018**, *68*, 597–608.
17. He, P.; Hao, J.K. Iterated two-phase local search for the colored traveling salesmen problem. *Engineering Applications of Artificial Intelligence* **2021**, *97*, 104018.
18. Zhou, Y.; Xu, W.; Fu, Z.H.; Zhou, M. Multi-neighborhood simulated annealing-based iterated local search for colored traveling salesman problems. *IEEE Transactions on Intelligent Transportation Systems* **2022**, *23*, 16072–16082.
19. Quan, C.; Guo, P. A local search method based on edge age strategy for minimum vertex cover problem in massive graphs. *Expert Systems with Applications* **2021**, *182*, 115185.
20. Andrade, D.V.; Resende, M.G.; Werneck, R.F. Fast local search for the maximum independent set problem. *Journal of Heuristics* **2012**, *18*, 525–547.
21. Cai, S.; Su, K.; Sattar, A. Local search with edge weighting and configuration checking heuristics for minimum vertex cover. *Artificial Intelligence* **2011**, *175*, 1672–1696.
22. Wang, Y.; Cai, S.; Yin, M. Local search for minimum weight dominating set with two-level configuration checking and frequency based scoring function. *Journal of Artificial Intelligence Research* **2017**, *58*, 267–295.
23. Cai, S.; Su, K.; Luo, C.; Sattar, A. NuMVC: An efficient local search algorithm for minimum vertex cover. *Journal of Artificial Intelligence Research* **2013**, *46*, 687–716.
24. Richter, S.; Helmert, M.; Gretton, C. A stochastic local search approach to vertex cover. annual conference on artificial intelligence. Springer, 2007, pp. 412–426.
25. Cai, S.; Su, K.; Chen, Q. EWLS: A new local search for minimum vertex cover. Proceedings of the AAAI Conference on Artificial Intelligence, 2010, Vol. 24, pp. 45–50.
26. Cai, S. Balance between complexity and quality: Local search for minimum vertex cover in massive graphs. Twenty-Fourth International Joint Conference on Artificial Intelligence, 2015.
27. Ma, Z.; Fan, Y.; Su, K.; Li, C.; Sattar, A. Local search with noisy strategy for minimum vertex cover in massive graphs. PRICAI 2016: Trends in Artificial Intelligence: 14th Pacific Rim International Conference on Artificial Intelligence, Phuket, Thailand, August 22-26, 2016, Proceedings 14. Springer, 2016, pp. 283–294.
28. Ma, Z.; Fan, Y.; Su, K.; Li, C.; Sattar, A. Random walk in large real-world graphs for finding smaller vertex cover. 2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI). IEEE, 2016, pp. 686–690.
29. Cai, S.; Lin, J.; Luo, C. Finding a small vertex cover in massive sparse graphs: Construct, local search, and preprocess. *Journal of Artificial Intelligence Research* **2017**, *59*, 463–494.
30. Rossi, R.A.; Ahmed, N.K. The Network Data Repository with Interactive Graph Analytics and Visualization. AAAI, 2015.