Information Sciences Letters
An International Journal

http://dx.doi.org/10.18576/isl/100209

Shrink: An Efficient Construction Algorithm for Minimum Vertex Cover Problem

Wael Mustafa

Department of Computer Science, An-Najah National University , Nablus, Palestine

Received: 20 Jan. 2020, Revised: 12 Feb 2020, Accepted: 4 Mar. 2021
Published online: 1 May 2021

Abstract: The minimum vertex cover (VC) problem is to find a minimum number of vertices in an undirected graph such that every edge in the graph is incident to at least one of these vertices. This problem is a classical NP-hard combinatorial optimization problem with applications in a wide range of areas. Hence, there is a need to develop approximate algorithms to find a small VC in a reasonable time. This paper presents a new construction algorithm for the minimum VC problem. Extensive experiments on benchmark graphs show that the proposed algorithm is extremely competitive and complementary to existing construction algorithms for minimum VC problem.

Keywords: minimum vertex cover, construction algorithm, combinatorial optimization, NP-hard

1 Introduction

Graph is a basic data structure with with many operations and applications in various domains. Given an undirected graph G with a set of vertices V and a set of edges E, a vertex cover for G is a subset S of V, such that each edge in E is incident to at least one vertex in S. Given a graph G, the minimum vertex cover (VC) problem is to find a cover containing the least number of vertices. The minimum VC problem is a well-known combinatorial optimization problem. It has applications in a wide range of fields including sensor networks [1], parallel machine scheduling [2], financial networks [3], economics [4, 5], social networks [6], and biotechnology [7, 8]. This paper proposes an effective algorithm for minimum VC problem.

Minimum VC problem is NP-hard problem. Previous work related to the minimum VC problem focused on approximate methods to obtain "reasonable" vertex covers in an admissible time. Existing algorithms for minimum VC problem can be classified into two categories. The first category is known as construction methods, and the second category is known as local search methods [9].

Construction algorithms for minimum VC problem typically begin with an empty cover and repeatedly add vertices into the cover. A construction algorithm usually ends when all all edges in the graph become covered. Examples of common construction algorithms for minimum VC problem are maximal matching, greedy vertex [10], and edge greedy [11].

In practice, construction algorithms for minimum VC problem are mainly employed to obtain a "good" cover that will be used as a starting cover in local search methods. The structure of a local search method for minimum VC problem often starts with using a construct algorithm to obtain a VC. Then, a vertex is removed from the initial cover. Later, small modifications are applied to the vertex set until it becomes a VC. Examples of these modifications include insertion of a new vertex, deleting a vertex, and exchanging a vertex in the cover with a vertex outside the cover. Modifications to the vertex set continue until either a satisfactory cover is found or a specific period of execution time expires. [12–17].

This paper presents a new construction algorithm, Shrink, for the minimum VC problem. Extensive experiments on benchmark graphs show that Shrink is extremely competitive with existing constructive minimum VC algorithms. The remainder of the paper is as follows. Section two reviews existing construction algorithms for minimum VC problem. Section three presents the proposed algorithm. Section four provides the results of applying the proposed algorithm to

* Corresponding author e-mail: wamustafa@yahoo.com

benchmark graphs and discusses the results. Finally, section five concludes the paper.

2 Background

An undirected graph $G = (V,E)$, where $V = \{v_1, v_2, ..., v_n\}$ is a set of vertices and $E = \{e_1, e_2, ..., e_m\}$ is a set of edges. An edge connecting vertex v_i with vertex v_j is represented by the set $\{v_i, v_j\}$ and this edge is said to be incident to v_i and v_j. For a vertex v in G, the neighborhood of v, denoted by $N(v)$, is the set containing all vertices adjacent to v, i.e., $N(v) = \{u \in V | \{u,v\} \in E\}$. The degree of a vertex v is the number of edges incident to v. $G \backslash T$ denotes a sub-graph obtained from G by removing all vertices of T and all edges with at least one vertex in T. A vertex cover of graph G is a subset S of V such that each edge in E is incident to at least one vertex in S. An edge that is incident to a vertex in S is said to be covered by S. The minimum VC problem is to find a VC with minimum number of vertices.

Example 1. Figure 1 shows an undirected graph $G = (V,E)$, where $V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$ and $E = \{\{v1, v2\}, \{v_1, v_3\}, \{v_1, v_4\}, \{v_1, v_5\}\}, \{v_5, v_6\}, \{v_2, v_3\}, \{v_2, v_5\}\}$. A minimum vertex cover for G is $\{v_1, v_2, v_6\}$.

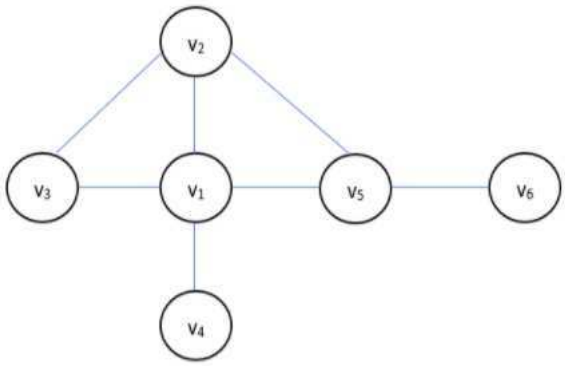

Fig. 1: Undirected graph G for example 1 .

A common and simple construction method for VC problem is to obtain a matching between verities in the graph that covers all edges. The vertices used in the matching are returned as a solution. In this paper, we call this algorithm Match. For a graph $G = (V,E)$, Match begins with initializing vertex cover C to the empty set. Then, it visits every edge $e \in E$; if e is not incident to a vertex in C, both endpoint vertices of e are added into C.

The pseudo code of Match algorithm is shown in Algorithm 1. Assuming that the graph is represented as a list of edges, the complexity of Match algorithm is O(m), where m is the number of edges [11]. This algorithm is often efficient with respect to execution time. However, in most cases the algorithm returns large covers compared to other construction algorithms.

Algorithm 1: Match algorithm for minimum VC problem .

```
Input: graph G = (V,E)
Output: vertex cover of G
1  C:=∅ ;
2  foreach edge e={v_i,v_j} ∈ E do
3  |  if e is uncovered by C then
4  |  |  add v_i and v_j to C;
5  |  end
6  end
7  return C;
```

Another known construction algorithm for minimum VC problem is a greedy method based on the gain in number of edges being covered by the selected vertices [10]. It is usually used as a construction algorithm to produce the starting cover for local search methods of minimum VC problem [12, 15, 16].

In this paper, the algorithm is called DegreeGreedy. To obtain a minimum VC for a graph $G = (V,E)$, DegreeGreedy first initializes vertex set C to the empty set. Then, the algorithm repeats the following until every edge in E becomes covered by C: choose the vertex v not currently in C that has the most uncovered edges and add v into C. Ties between vertices with equal number of uncovered edges are broken randomly. The pseudo code of DegreeGreedy is shown in Algorithm 2. The time complexity of DegreeGreedy is $O(n^2)$, where n is the number of vertices in the graph [11].

Algorithm 2: DegreeGreedy algorithm for minimum VC problem .

```
Input: graph G = (V,E)
Output: vertex cover of G
1  C:=∅
2  foreach vertex v in V do
3  |  compute degree[v];
4  end
5  k:=the vertex in V with maximum degree;
6  while degree[k]!=0 do
7  |  add k to C;
8  |  degree[k]=0;
9  |  foreach vertex v neighboring k do
10 |  |  degree[v] - -;
11 |  end
12 |  k:=the vertex in V with maximum degree;
13 end
14 return C;
```

The third vertex cover construction algorithm, EdgeGreedy, was proposed in [11]. The pseudo-code of EdgeGreedy is given in Algorithm 3. EdgeGreedy Algorithm consists of two stages. In the first stage, the algorithm starts with an empty cover. Then, the algorithm scans all edges in the graph and whenever it finds an uncovered edge $\{v_i, v_j\}$, the vertex that has a higher degree in $\{v_i, v_j\}$ is inserted into the cover. In case v_i and v_j are equal in the degree, the vertex with lower index (first vertex) is chosen. A VC is created at the end of the first stage of the algorithm. In the second stage, the loss value of each vertex in the cover is calculated. After that, the algorithm scans the vertices in the cover. Whenever the algorithm finds a vertex v with a loss value of zero, it deletes v from the cover and updates the loss values of all vertices in N[v]. The time complexity of EdgeGreedy is O(m), where m is the number of edges [11] .

Algorithm 3: EdgeGreedy algorithm for minimum VC problem.

```
   Input: graph G = (V,E)
   Output: vertex cover of G
1  C:=∅
2  foreach e = {v_i, v_j} ∈ E do
3  |  if e is uncovered by C then
4  |  |  add the vertex in e with higher degree to C;
5  |  end
6  end
7  foreach v ∈ C do
8  |  loss(v):=0;
9  end
10 foreach e = {v_i, v_j} ∈ E do
11 |  if only one of v_i and v_j belongs to C then
12 |  |  v:=the vertex in e that belongs to C;
13 |  |  loss(v)++;
14 |  end
15 end
16 foreach v ∈ C do
17 |  if loss(v)=0 then
18 |  |  C:=C\{v} ;
19 |  |  Update loss of vertices neighboring v;
20 |  end
21 end
22 return C;
```

3 The Proposed Minimum VC Algorithm

This section presents the proposed construction algorithm for minimum VC problem. For convenience, the algorithm is called Shrink. The algorithm first initializes the cover to the entire set of vertices. Then, it reduces the cover by deleting vertices whose removal does not reduce the number of covered edges. The pseudo code of the algorithm is shown in Algorithm 4. The algorithm associates a *count-value* $\in \{1,2\}$ with every edge in the graph. A *count-value* of 2 indicates that the edge is incident to two vertices in the current vertex cover C, and a *count-value* of 1 indicates that the edge is incident to only one vertex in C.

Algorithm 4: Shrink algorithm for minimum VC problem.

```
   Input: graph G = (V,E)
   Output: vertex cover of G
1  C:=V
2  foreach e ∈ E do
3  |  count-value[e]=2;
4  end
5  Compute the degree of all vertices in V;
6  N:= the list of vertices in V sorted in ascending order of
     degree;
7  for i :=1 to |V| do
8  |  if every edge e incident to vertex N[i] has
   |    count-value[e]=2 then
9  |  |  C:=C\{N[i]};
10 |  |  foreach edge e incident to N[i] do
11 |  |  |  count-value[e]=1;
12 |  |  end
13 |  end
14 end
15 return C;
```

In line 1 of the pseudo code, Shrink initializes the vertex cover C to include the entire set of vertices V. Lines 2-4 initialize the *count-value* of every edge in E to 2 since every edge is incident to two vertices in the cover. Line 5 computes the degree of all vertices in the graph. Line 6 sorts the vertices of the graph in ascending order of degree and stores the result in the list of vertices N. The loop in lines 7-14 removes vertices from C whenever possible. Vertices with smaller degrees are checked for removal earlier. So, if the graph contains isolated vertices with degree 0, these vertices are removed from C in early iterations. Removal of these vertices does not cause any edge to be uncovered. In later iterations the algorithm checks vertices with higher degrees for possibility of removal from C. Whenever the algorithm finds that a vertex v with all incident edges having *count-value* 2, it removes v from the cover and the count values of edges incident to v are all set to 1. If the count-vale of an edge $\{v_i, v_j\}$ becomes 1, the two vertices v_i and v_j will be retained in the cover in later iterations. The algorithm starts with the removal of vertices with smaller degrees since removing these vertices has smaller effect on the number of possibilities to cover edges in the graph. Whenever choosing between vertices with equal degrees for removal from the cover, Shrink always selects the first vertex.

Example 2. To illustrate how the algorithm works, we apply Shrink to the graph in figure 1. First, the cover C is

initialized to $V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$. Then, the *count-value* of all edges is set to 2. Next, sorting the vertices in ascending order of degree yields the list $N = [v_4, v_6, v_3, v_2, v_5, v_1]$. Ties between v_4, v_6 and between v_2, v_5 are broken by selecting the vertex with smaller number. So, v_4 comes before v_6 and v_2 comes before v_5 in the list N. The algorithm then goes through each vertex in N in order and examines if the vertex can be removed from C without causing edges to be uncovered. At v_4, the only incident edge to this vertex has *count-value* 2, v_4 is removed from the cover, and the count-value of the edge $\{ v_4, v_1 \}$ is set to 1. The graph at this stage is shown in Figure 2.

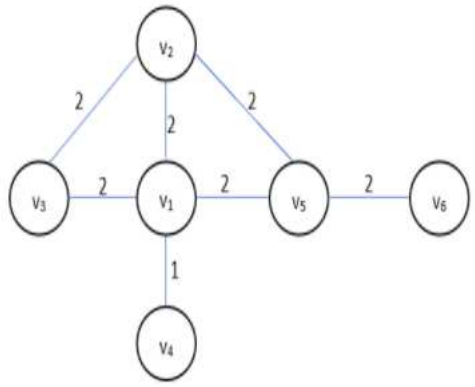

Fig. 2: The graph G after removal of v_4 from the cover.

Next, v_6 is examined for removal from the cover. Again, the edge incident to this vertex has a *count-value* of 2, so v_6 is also removed from the cover, and the count-value of the edge $\{v_6, v_5\}$ is set to 1. Then, v_3 is checked for removal. Since the count-value of the two edges incident to v_3 are both 2, v_3 is removed from the cover and the *count-value* of the two edges incident to *v3* are set to 1. At this stage, the the cover $C = \{v_1, v_2, v_5\}$ and the graph is shown in Figure 3.

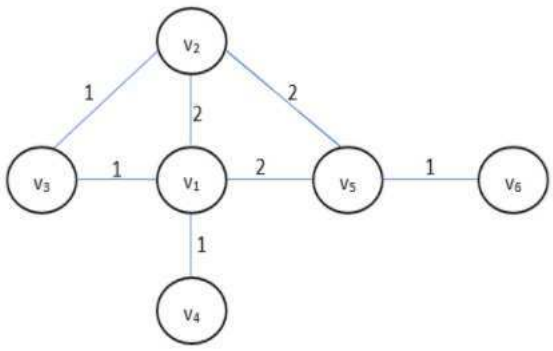

Fig. 3: The graph G after removal of v_4, v_6, v_3 from the cover.

At v_2, the *count-value* of the edge $\{v_2, v_3\}$ is 1, so v_2 is retained in the cover. At v_5, the *count-value* of the edge $\{v_5, v_6\}$ is 1, so v_5 also remains in the cover. Finally, at v_1 there are two edges incident to this vertex that have *count-value* equal to 1, and v_1 is retained in the cover. The cover output by the algorithm is $C = \{v_1, v_2, v_5\}$.

The algorithm always produces a minimum cover. This can be proved as follows. At line 1, C is initialized to all the vertices in V. By graph definition, each edge in E is incident to some vertex in V, so the initial cover C created at step 1 is a cover for G. The loop in the lines 7-14 reduces C to a minimum cover. The output cover can be proved to be a minimum cover by contradiction: Assume output cover is not minimum. This implies that there is a vertex v in the output cover C that can be removed and C remains a cover. Since v can be removed from C, all edges incident to v has *count-value* equal to 2. However, such a vertex will be removed from the cover in the i'th iteration of the loop, where i is the index of vertex v in the list N. Hence, vertex v cannot be in the output cover C. This completes the proof by contradiction that the algorithm returns a minimum cover.

Following is the worst-case time complexity analysis for the proposed algorithm. In the analysis n represents the number of vertices and m represents the number of edges in the graph. The statement in line 2 can be implemented in complexity O(m). In the worst-case, when the graph is nearly complete and every edge is connected to most other edges, $m \cong \frac{n(n-1)}{2}$ and the complexity $O(m) = O(\frac{n(n-1)}{2}) = O(n^2)$. Computing the degrees of all vertices in the graph, line 5, can be implemented in $O(n+m)$ by starting with an empty graph (all the degrees are zero), complexity $O(n)$, and then adding the edges one by one, complexity $O(m)$; when adding an edge to the graph only two vertices increment their degrees. Thus, The complexity of computing degree of all vertices $O(n+m) = O(n+\frac{n(n-1)}{2}) = O(n^2)$. Sorting the vertices in ascending order of degree, line 6, can be implemented in $O(n\log(n))$. The for loop in line 7 executes n iterations. In the i'th iteration, each of the two statements in line 8 and line 10 requires visiting $degree(i)$ edges. Hence, the loop in lines 7-14 complexity is $\sum_{u \in V}(degree(u) + degree(u)) = 2\sum_{u \in V}(degree(u)) = 2(2m) = O(m) = O(\frac{n(n-1)}{2}) = O(n^2)$. Therefore, the time complexity of the proposed algorithm is $O(n^2)$.

4 Results and Discussion

This section reports and compares the results of the proposed Shrink algorithm with the three existing minimum VC construction algorithms: Match, DegreeGreedy, and EdgeGreedy.

The four algorithms were implemented as functions in a single program in visual C++ 2019. The resulting code was executed on an HP computer with an i7 CPU,

Table 1: Comparing the four construction algorithms for minimum VC problem.

			Match		Shrink		DegreeGreedy		EdgeGreedy	
Graph	$\|N\|$	$\|E\|$	size	time	size	time	avg size	avg time	size	time
3elt-dual	9000	13278	8914	2.937	**4804**	6.283	4857	3.834	**4804**	5.859
bio-celegans	453	2025	398	0.006	257	0.018	**255**	0.012	259	0.023
bio-diseasome	516	1188	400	0.008	**285**	0.022	**285**	0.013	**285**	0.02
bio-dmela	7393	25569	4076	1.599	2716	4.484	**2667**	2.267	2717	3.914
bio-yeast	1458	1948	800	0.066	**462**	0.181	464	0.868	**462**	0.153
brock200-1	200	14834	200	0.001	**196**	0.003	**196**	0.003	**196**	0.005
brock200-2	200	9876	198	0.001	**192**	0.003	193	0.003	**192**	0.005
brock800-1	800	207505	798	0.018	**792**	0.049	793	0.047	793	0.074
brock800-2	800	208166	800	0.019	**792**	0.051	794	0.048	793	0.08
brock800-3	800	207333	800	0.019	**793**	0.05	794	0.045	**793**	0.077
brock800-4	800	207643	800	0.018	**793**	0.048	**793**	0.045	**793**	0.077
C100-9	1000	450079	1000	0.035	997	0.078	**996**	0.065	997	0.131
C152-9	125	6963	124	0.004	**121**	0.001	122	0.001	**121**	0.019
C2000-5	2000	999836	1998	0.153	**1988**	0.326	1989	0.292	**1988**	0.474
C2000-9	2000	1799532	2000	0.147	**1995**	0.316	1996	0.258	1996	0.546
C250-9	250	27984	248	0.013	246	0.004	**245**	0.004	247	0.008
C4000-5	4000	4000268	3998	0.65	3987	1.295	3990	1.148	**3986**	1.968
C500-9	500	112332	500	0.007	**496**	0.018	497	0.017	**496**	0.032
ca-CSphd	1882	1740	1044	0.104	**553**	0.318	555	0.147	**553**	0.251
ca-Erdos992	6100	7515	594	0.797	**461**	3.244	**461**	1.206	**461**	2.394
ca-GrQc	4158	13422	3214	0.567	**2214**	1.339	2219	0.802	**2214**	1.285
ca-HepPh	11204	117619	9088	4.414	**6562**	9.444	6574	5.981	6565	9.255
ca-netscience	379	914	300	0.004	**214**	0.116	**214**	0.073	**214**	0.012
DSJC1000-5	1000	249826	998	0.037	991	0.079	**990**	0.073	**990**	0.118
DSJC500-5	500	62624	498	0.007	**490**	0.021	491	0.019	**490**	0.029
EX1	560	4368	560	0.01	**460**	0.023	468	0.019	464	0.027
EX2	560	4368	560	0.012	480	0.03	**456**	0.025	474	0.032
EX4	2600	35880	2600	0.352	**2263**	0.628	2282	0.537	2279	0.749
EX5	6545	147840	6542	1.489	**5959**	2.835	5965	2.551	5983	3.531
EX6	6545	147840	6542	1.456	5982	2.778	**5974**	2.508	6003	3.465
fe-4elt2	11143	32818	10886	4.326	8127	8.845	8155	6.552	**8119**	9.27
hamming10-2	1024	518656	1024	0.027	**1022**	0.077	**1022**	0.068	**1022**	0.134
hamming10-4	1024	434176	1024	0.029	**1016**	0.072	**1016**	0.066	**1016**	0.125
ia-email-univ	1133	5451	814	0.077	614	0.102	**605**	0.062	615	0.096
ia-enron-only	143	623	126	<0.001	**87**	0.002	88	0.001	**87**	0.002
ia-fb-messages	1266	6451	932	0.937	**592**	2.692	595	1.342	**592**	2.729
ia-infect-dublin	410	2765	372	0.006	299	0.013	**298**	0.01	300	0.014
ia-infect-hyper	113	2196	108	<0.001	**93**	0.001	**93**	0.001	**93**	0.001
ia-reality	6809	7680	114	0.916	**81**	4.373	**81**	1.4	**81**	2.951
inf-power	4941	6594	3736	0.807	**2269**	1.951	2277	1.085	2271	1.767
MANN-a27	378	70551	376	0.003	376	0.01	**375**	0.009	376	0.018
MANN-a45	1035	533115	1034	0.028	1033	0.076	**1032**	0.068	1033	0.138
MANN-a9	45	918	44	<0.001	43	<0.001	**42**	<0.001	43	<0.001
rt-retweet	96	117	60	<0.001	**33**	<0.001	**33**	<0.001	**33**	<0.001
rt-twitter-copen	761	1029	422	0.021	**238**	0.055	**238**	0.023	**238**	0.041
socfb-CMU	6621	249959	6290	1.597	5082	3.105	**5067**	2.398	5090	3.6
socfb-Duke14	9885	506437	9422	3.568	7827	6.975	**7807**	5.371	7838	7.999
socfb-MIT	6402	251230	6014	1.471	4744	2.963	**4732**	2.209	4756	3.303
socfb-Stanford3	11586	568309	10796	4.888	8671	9.559	**8626**	7.2	8696	10.666
socfb-UCSB37	14917	482215	14116	8.638	**11488**	15.676	11494	11.998	11510	18.284
soc-dolphines	62	159	50	<0.001	**35**	0.001	**35**	0.001	**35**	0.001
soc-karate	34	78	22	<0.001	**14**	<0.001	**14**	<0.001	**14**	<0.001
soc-wike-vote	889	2914	650	0.02	413	0.697	**411**	0.037	414	0.058
tech-routers-rf	2113	6632	1376	0.145	**803**	0.378	807	0.193	**803**	0.328
tech-WHOIS	7476	56943	3380	1.634	2304	4.794	**2296**	2.288	2305	4.052
web-BerkStan	12305	19500	8432	4.645	5567	12.117	**5537**	6.624	5565	10.737

Table 2: Comparing the four construction algorithms for minimum VC problem.

			Match		Shrink		DegreeGreedy		EdgeGreedy	
Graph	$\|N\|$	$\|E\|$	size	time	size	time	avg size	avg time	size	time
web-edu	3031	6474	2818	0.35	**1451**	0.742	**1451**	0.425	**1451**	0.683
web-google	1299	2773	478	0.051	**499**	0.141	**499**	0.077	**499**	0.12
web-indochina-2004	1358	47606	9720	4.344	**7366**	9.589	**7366**	6.51	7367	9.45
web-polblogs	643	2280	406	0.012	**246**	0.04	**246**	0.018	**246**	0.031
web-spam	4767	37375	3566	0.761	2351	1.814	**2339**	1.036	2352	1.707
web-webbase-2001	16062	25593	4284	6.339	**2671**	22.927	2679	8.976	2674	17.105

running Windows 10; 2.2 GHz speed; and an eight GB of main memory. The four algorithms were applied to 62 undirected unweighted graph instances from the NetworkRepository [18], which is often used in the literature to benchmark algorithms for combinatorial graph problems. Instant graphs were represented in memory as adjacency matrices.

Tables 1 and 2 contain the size of the output cover and execution time in seconds for each algorithm. Execution time is calculated based on high_resolution_clock::now() C++ function. To account for breaking ties randomly between vertices with an equal degree in DegreeGreedy, the algorithm was applied to every graph 10 times using 10 different seeds for the random function used to implement the random selection between vertices. The average cover size, rounded to the nearest integer, and the average execution time in seconds over the 10 executions is reported. The best cover size among the four algorithms is shown in bold face for each graph. The tables also show the size of the graphs where $|N|$ is the number of vertices and $|E|$ is the number of edges.

Some observations about the quality of the output cover and execution time of the four algorithms are as follows.

1. The three algorithms: Shrink, DegreeGreedy, and EdgeGreedy always find better covers than Match algorithm and these three algorithms are almost competitive. Furthermore, these algorithms are often complementary; for a specific graph there is a good chance that one of the three algorithms produces the best cover. In numbers, when Shrink is compared to the three other algorithms, it produces the smallest cover for 40 graphs. DegreeGreedy finds the best cover for 37 graphs. EdgeGreedy finds the best cover for 33 graphs.
2. Match has least execution time in 61 out of 62 graphs.
3. When comparing execution time of the three remaining algorithms: DegreeGreedy, EdgeGreedy, and Shrink, DegreeGreedy has least execution time among the three algorithms in 61 graphs, Shrink has least execution time in 10 graphs, and EdgeGreedy has least execution time in 6 graphs.
4. When comparing execution time of Shrink against EdgeGreedy, Shrink finished faster in 32 instances, while EdgeGreedy finished faster in 15 instances. In the remaining 15 instances, the two algorithms have similar execution time.
5. Overall, Shrink balances between quality of output vertex cover and execution time. It is extremely competitive and complementary to DegreeGreedy. Shrink is better EdgeGreedy with respect to both quality of output and execution time. Shrink is better than Match with respect to quality of output cover.

5 Conclusion

This paper presented a new construction algorithm for minimum VC problem. Extensive experiments on benchmark undirected graphs show that the proposed algorithm is highly competitive to existing algorithms. Furthermore, the proposed algorithm balances between solution quality and execution time. A future work is to investigate the effectiveness of integrating the algorithm into local search algorithms for minimum VC problem.

Competing interests

The authors declare that they have no competing interests.

References

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

[2] W. Hong, and Z. Wang. Improved Approximation Algorithm for the Combination of Parallel Machine Scheduling and Vertex Cover, *Int. J. Found. Comput. Sci*, **28**, 977–992 (2017).

[3] V. Boginski, S. Butenko, and P. Pardalos. Mining market data: A network approach, *Comput. Oper. Res.*, **33**, 3171–3184 (2006).

[4] Q. Wu, and J. Hao. A review on algorithms for maximum clique problems, *Eur. J. Oper. Res*, **242**, 693–709 (2015).

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

[6] T. Yadav, K. Sadhukhan, and A. Rao. Approximation algorithm for n-distance minimal vertex cover problem, *CoRR, abs/1606.02889*, (2016).

[7] A. Hossain, E. Lopez, S. Halper, D. Cetnar, A. Reis, D. Strickland, E. Klavins, and H. Salis. Automated design of thousands of nonrepetitive parts for engineering stable genetic systems, *Nature Biotechnology*, **38**, 1466-1475 (2020).
[8] A. Reis, S. Halper, G. Vezeau, D. Cetnar, A. Hossain, P. Clauer, and H. Salis. Simultaneous repression of multiple bacterial genes using nonrepetitive extra-long sgRNA arrays, *Nature Biotechnology*, **37**, 1294–1301 (2019).
[9] H. Hoos, and T. Stützle. *Stochastic Local Search: Foundations and Applications*, Morgan Kaufmann, (2004).
[10] C. Papadimitrious, and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*, Prentice Hall, (1982).
[11] S. Cai, J. Lin, and C. Luo. Finding A Small Vertex Cover in Massive Sparse Graphs: Construct, Local Search, and Preprocess, *Journal of Artificial Intelligence Research*, **59**, 463-494 (2017).
[12] S. Richter, M. Helmert, and C. Gretton. A stochastic local search approach to vertex cover, *In Proceedings of KI 2007*, 412–426 (2007).
[13] D. Andrade, M. Resende, and R. Werneck. Fast local search for the maximum independent set problem, *In Workshop on Experimental Algorithms*, 220–234 (2008).
[14] W. Pullan. Optimisation of unweighted/weighted maximum independent sets and minimum vertex covers, *Discrete Optimization*, **6**, 214–219 (2009).
[15] S. Cai, K. Su, K., and A. Sattar. Local search with edge weighting and configuration checking heuristics for minimum vertex cover, *Artif. Intell.*, **175**, 1672–1696 (2011).
[16] S. Cai, K. Su, K., and A. Sattar. NuMVC: An efficient local search algorithm for minimum vertex cover, *J. Artif. Intell. Res. (JAIR)*, **46**, 687–716 (2013).
[17] W. Luzhi, S. Hu, M. Li, and J. Zhou. An Exact Algorithm for Minimum Vertex Cover Problem, *Mathematics*, **7**, 603 (2019), doi:10.3390/math7070603.
[18] R. Rossi and N. Ahmed. The Network Data Repository with Interactive Graph Analytics and Visualization, *In Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, Austin, TX, USA*, 25–30 (2015).