Vertex Cover Problem

Exact and Approximation Algorithms

Iftekhar Hakim Kaowsar
Student ID : 1705045

Apurba Saha
Student ID : 1705056

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology

July 27, 2021

Contents

1 Introduction **3**

2 Problem Definition **3**

3 Exact Algorithms **4**
3.1 Brute Force Algorithm . . . 4
3.1.1 Overview . . . 4
3.1.2 Pseudocode . . . 4
3.1.3 Time Complexity . . . 5
3.2 Parameterized Vertex Cover . . . 5
3.2.1 Overview . . . 5
3.2.2 Pseudocode . . . 6
3.2.3 Time Complexity . . . 7

4 Approximation Algorithm **8**
4.1 Overview . . . 8
4.2 Pseudocode . . . 8
4.3 Time Complexity . . . 10
4.4 Space Complexity . . . 10
4.5 Proof of Factor 2 Approximation . . . 11

5 Comparison **12**

6 Room for Improvement **13**

7 Conclusion **13**

1 Introduction

This report explains the well-known optimization problem Vertex Cover. In computer science, the problem of finding a minimum vertex cover is a classical optimization problem. In this report, we have shown two exact algorithms (one is brute-force and another is parameterized vertex cover). We have also shown an approximation algorithm for finding near optimal solution as we cannot use exact algorithms in real world problems.

2 Problem Definition

Suppose $G(V, E)$ is a graph, where V is the set of vertices and E is the set of edges. **Vertex Cover** of graph G means a subset $V' \subseteq V$, so that for every edge $(u, v) \epsilon E$, either $u \epsilon V'$ or $v \epsilon V'$ or both are true.
Vertex Cover Problem is about choosing the subset with minimum possible size, i.e. choosing $V' \subseteq V$ with minimum possible size where V' is a vertex cover of G.

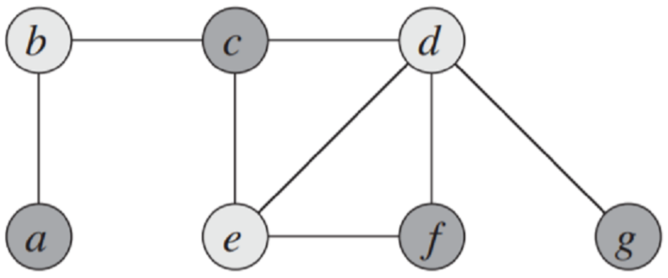


Figure 1: Vertex Cover

From Figure 1, vertices set $\{b, d, e\}$ comprises minimum vertex cover of the graph. However, set $\{a, c, e, f, g\}$ is another vertex cover, although it is not optimal. Again, set $\{a, b, c, f, g\}$ is not a vertex cover at all despite having a larger size than optimal set.

A real world scenario of this problem can be described with an example of an area, where we need to set cameras to cover different roads. Graph edges will represent the roads and vertices will represent the end points. Now, finding minimum number of cameras is similar to vertex cover problem. Another example can be a museum site, where we need to set minimum number of guards.

Decision version: The decision version of the problem is about deciding whether a graph has a vertex cover of size at most k. If we can solve the decision version of the problem, we can solve the original problem (optimization version) too.

Vertex cover problem is NP-complete. We do not expect a polynomial-time solution to solve it. This fact increases the weight of the problem.

3 Exact Algorithms

As we have already stated, there is no polynomial time solution known which can solve this problem. We can run a complete brute force to solve for a given graph. There are other algorithms too which just make optimizations to the complexity. We will analyze the brute force algorithm and the parameterized algorithm to solve the decision version of the problem.

3.1 Brute Force Algorithm

3.1.1 Overview

From the problem description, the brute force algorithm for vertex cover problem is too obvious. Let us go through its details.
Let the graph has n vertices and we need to decide whether there is vertex cover of size at most k. If we can, we will also find the vertex cover set.
The outline of the algorithm is —

- If graph has more than $(n-1)k$ edges, return *no* decision.
- Assume decision is *no*.
- Check for each possible subset of the vertices with size at most k as a vertex cover. If it is suitable, make the decision *yes* and return this subset.

We observe that a selection of k nodes can cover at most $(n-1)k$ edges. That is why, if graph has more than $(n-1)k$ edges, we can directly make our decision as *no*.

3.1.2 Pseudocode

Algorithm 1: Brute-force $VC(G,k)$

```
  Input: Graph $G$ and integer $k$
  Output: Returns vertex cover set if it exists
1 if $G.E.size > (n-1)k$ then
2     return "no"
3 end
4 for each $U \subseteq G.V$ with $U.size \leq k$ do
5     if $U$ is vertex cover then
6         return $U$
7     end
8 end
9 return "no"
```

3.1.3 Time Complexity

Let us assume adjacency list has been used to store the graph. Also, let n is the number of vertices and m is the number of edges. Now,

- Number of possible subset of size k from a set of size n is $\binom{n}{k}$.
- Testing whether a single subset covers the graph takes $O(m)$. We can rewrite it as $O(nk)$. Because if $m > (n-1)k$, there will never be a vertex cover of size k.

So, overall time complexity is $O\left(nk\binom{n}{k}\right) = O\left(kn^{k+1}\right)$. We note that it is exponential of n or the number of vertices.

3.2 Parameterized Vertex Cover

3.2.1 Overview

Now, we explore another algorithm which can solve our problem accurately. We call it *Parameterized Vertex Cover (PVC)*. It will make a little improvement over the time complexity of brute-force algorithm. But it is still exponential. After analyzing it, we will make a comparison between brute-force algorithm and PVC.

Let $G(V, E)$ is a graph where V is the set of vertices and E is the set of edges. (u, v) denotes an edge where u and v are the endpoints. We need to find out if there is a vertex cover of size at most k.
Key Observation:
If edge $e_1 = (u, v)$ is an edge of graph G, any vertex cover must select at least one of u and v. That means u or v must be one the selected k vertices.
Let us assume our final decision is *yes* and we have vertex u in our vertex cover set. This implies there is vertex cover of size at most $k-1$ in graph $G-u$. By term $G-u$, we mean the graph after excluding vertex u and its incident edges from graph G.

Without loss of generality, this observation is correct in case we have vertex v in our vertex cover set. Hence, we get our recurrence relation.

$$decision(G, k) = decision(G-u, k-1) \text{ or } decision(G-v, k-1)$$

where (u, v) is any edge of graph G.
The base case is,

$$decision(G, 0) = \begin{cases} true & \text{if G has no edge} \\ false & \text{otherwise} \end{cases}$$

Just like before, we can direct make our decision as *no*, if graph has more than $(n-1)k$ edges at any moment.

Figure 2 shows recursion tree of PVC for an example graph. It is self-explanatory. Every time we break the graph by erasing a vertex. Once we reach the leaf level where our parameter or k equals 0. If we can make *yes* decision at a leaf, we can go upward to the root of recursion tree to get the vertex cover set. In Figure 2, set $\{u, y, v\}$ or $\{v, x, w\}$ are the possible vertex cover set.

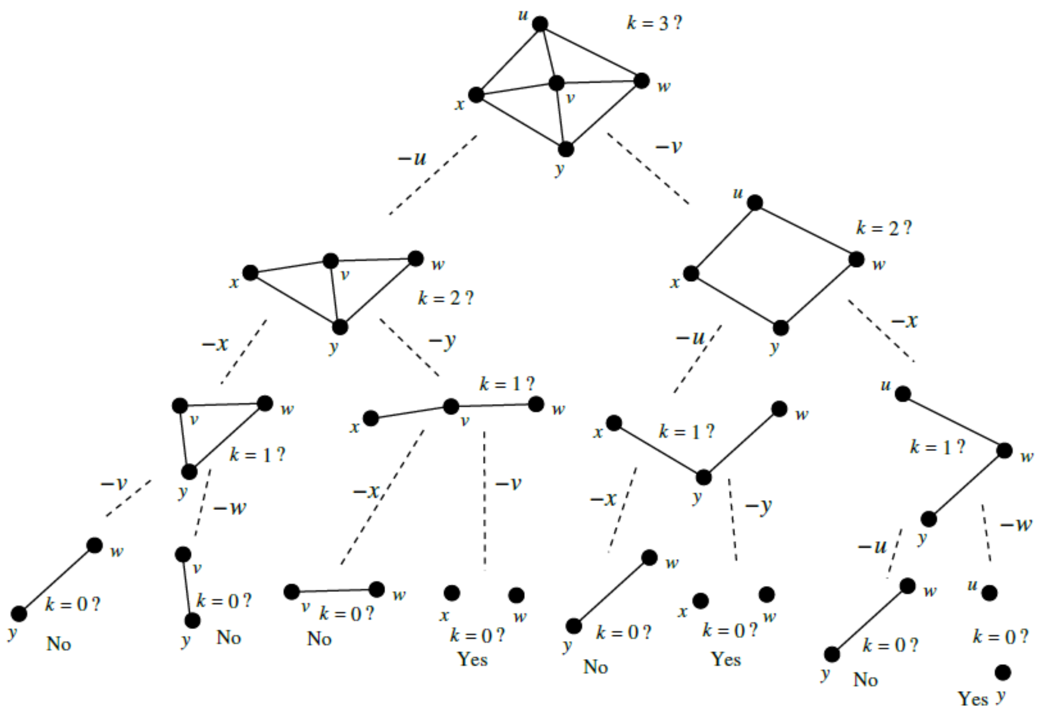


Figure 2: Parameterized Vertex Cover

3.2.2 Pseudocode

Algorithm 2: Parameterized-Algorithm $PVC(G, k)$

```
    Input: Graph $G$ and integer $k$
    Output: Returns vertex cover set if it exists
 1  if $G.E.size > (n-1)k$ then
 2      return "no"
 3  end
 4  if $G.E.size = 0$ then
 5      return $\emptyset$
 6  end
 7  if $k = 0$ then
 8      return "no"
 9  end
10  $(u, v) \leftarrow$ first element of $G.E$
11  if $PVC(G-u, k-1) \neq$ "no" then
12      $S \leftarrow$ the return set of $PVC(G-u, k-1)$
13      return $S \cup u$
14  end
15  if $PVC(G-v, k-1) \neq$ "no" then
16      $S \leftarrow$ the return set of $PVC(G-v, k-1)$
17      return $S \cup v$
18  end
19  return "no"
```

3.2.3 Time Complexity

To calculate $PVC(G, k)$, we recur to two branches, each requires rebuilding the adjacency list. Let $T(n, k)$ be the time required for n vertices and parameter k. However, rebuilding the adjacency list takes $O(|E|)$ time, we can rewrite it as *constant* $\times\ kn$, because $|E| \leq (n-1)k$ for *yes* decision. We can solve the recurrence relation by iterative substitution method.

$$\begin{aligned}
T(n,k) &\leq 2T(n,k-1) + ckn \\
&< 2\{2T(n,k-2) + ckn\} + ckn \\
&= 4T(n,k-2) + 2ckn + ckn \\
&< 8T(n,k-3) + 4ckn + 2ckn + ckn \\
&= 2^3T(n,k-3) + 2^2ckn + 2^1ckn + 2^0ckn \\
&< 2^kT(n,0) + 2^{k-1}ckn + 2^{k-2}ckn + ... + 2^1ckn + 2^0ckn \\
&< 2^k + 2^kckn \\
&= O(2^kkn)
\end{aligned}$$

So, time complexity for the parameterized vertex cover problem is $O(2^kkn)$.
We see it is still exponential, not feasible with a large graph. However, it is much better than the brute-force algorithm. Brute-force algorithm's time complexity includes exponential of n or number of vertices, whereas PVC's time complexity includes exponential of 2.

4 Approximation Algorithm

4.1 Overview

Previously we have learned that vertex cover is an NP-complete problem. Even though (assuming $\mathbf{P} \neq \mathbf{NP}$) we can not hope for a polynomial time algorithm that always gets the best solution, we can develop polynomial time algorithms that produce a near optimal solution. In this section, we consider such an approximation algorithm for vertex cover problem.

The following approximation algorithm takes as input an undirected graph G and returns a vertex cover whose size is guaranteed to be no more than twice the size of an optimal vertex cover.

- First we pick a random edge say, (u, v).
- We know any vertex cover must have at least one endpoint of it, so we take both endpoints.
- Then we throw out all edges incident to either u or v.
- We keep repeating previous 3 steps until there are no uncovered edges left.

4.2 Pseudocode

Algorithm 3: APPROX-VERTEX-COVER

```
Input: Graph $G$
Output: Vertex cover set
1 $C = \emptyset$
2 $E' = G.E$
3 while $E' \neq \emptyset$ do
4     let, $(u, v)$ be an arbitrary edge of $E'$
5     $C = C \cup \{u, v\}$
6     remove from $E'$ every edge incident to either $u$ or $v$
7 end
8 return $C$
```

The variable C contains the vertex cover being constructed. Line 1 initializes C to the empty set. Line 2 sets E' to be a copy of the edge set $G.E$ of the graph. The loop of lines 3–7 repeatedly picks an edge (u, v) from E' , adds its endpoints u and v to C, and deletes all edges in E' that are covered by either u or v. Finally, line 8 returns the vertex cover C.

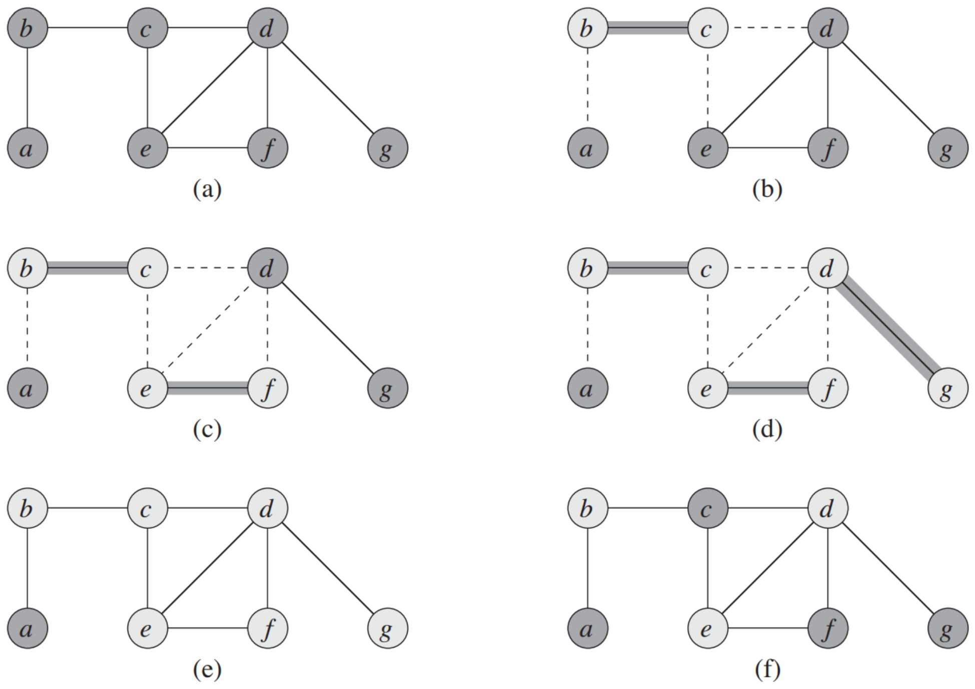


Figure 3: The simulation of APPROX-VERTEX-COVER. **(a)** The input graph G has seven vertices and eight edges. **(b)** The edge (b, c), shown bold, is the first edge chosen by APPROX-VERTEX-COVER. Vertices b and c, shown lightly shaded, are added to the vertex cover being created. Edges (a, b), (c, e), and (c, d), shown dashed, are removed since they are now covered by some vertex in C. **(c)** Edge (e, f) is chosen; vertices e and f are added to C. **(d)** Edge (d, g) is chosen; vertices d and g are added to C. **(e)** The set C, which is the vertex cover produced by APPROX-VERTEX-COVER, contains the six vertices $\{b, c, d, e, f, g\}$. **(f)** The optimal vertex cover for this problem contains only three vertices: $\{b, d, e\}$.

4.3 Time Complexity

1. If we consider adjacency lists to represent Graph G
   - It takes $O(1)$ time to execute Line 1.
   - Line 2 copies all the edges of the graph. So it takes $O(|V|+|E|)$ time to execute Line 2.
   - From Line 3 to 7 we have a loop which runs until there are no edges left. So it takes $O(|V|+|E|)$ time to execute Line 3-7.
   - Line 8 returns the answer. It takes $O(|V|)$ time to execute.

   So, the running time of this algorithm is $O(|V|+|E|)$

2. If we consider adjacency matrix to represent Graph G
   - It takes $O(1)$ time to execute Line 1.
   - Line 2 copies all the edges of the graph. So it takes $O\left(|V|^2\right)$ time to execute Line 2.
   - From Line 3 to 7 we have a loop which runs until there are no edges left. So it takes $O\left(|V|^2\right)$ time to execute Line 3-7.
   - Line 8 returns the answer. It takes $O(|V|)$ time to execute.

   So, the running time of this algorithm is $O\left(|V|^2\right)$

4.4 Space Complexity

1. The space complexity of this algorithm is $O(|V|+|E|)$ using adjacency list representation.
2. The space complexity of this algorithm is $O\left(|V|^2\right)$ using adjacency matrix representation.

4.5 Proof of Factor 2 Approximation

Theorem 1 *The APPROX-VERTEX-COVER is a polynomial time factor 2 approximation algorithm.*

Proof.
Let,

C denote the set of vertices that is returned by APPROX-VERTEX-COVER

C^* denote the set of vertices in the optimal vertex cover

K denote the set of edges picked by APPROX-VERTEX-COVER

Key Observations

- In the set of edges that were picked, no two share an endpoint.
- If the algorithm picked K edges, the vertex cover found has size $2K$.

From these observations, any vertex cover must have size at least K since it needs to have at least one endpoint of each of these edges. So, we have the lower bound

$$|C^*| \geq |K| \tag{1}$$

But, the vertex returned by the APPROX-VERTEX-COVER has size $2K$, yielding an exact upper bound on the size of the vertex cover returned

$$|C| = 2|K| \tag{2}$$

Combining Equations (1) and (2), we obtain

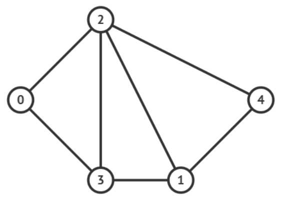
$$\begin{aligned} |C| &= 2|K| \\ &\leq 2|C^*| \end{aligned} \tag{3}$$

thereby proving the theorem.

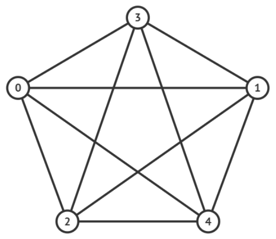
Interesting fact: Nobody knows any algorithm with approximation ratio 1.9. Best known is $2 - O\left(\frac{1}{\sqrt{\log n}}\right)$.

5 Comparison

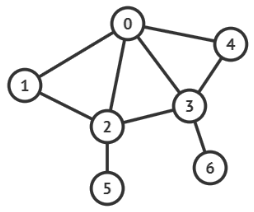
Below we show some graphs and compare the result of exact algorithm with approximation algorithm.



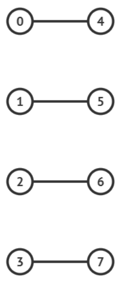
(a) Graph 1



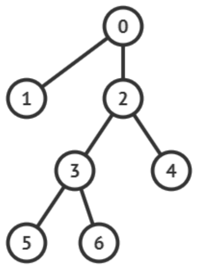
(b) Graph 2



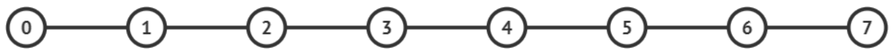
(c) Graph 3



(d) Graph 4



(e) Graph 5



(f) Graph 6

Graph No.	Exact Algorithm		Approximation Algorithm	
	Probable Vertex Cover	Cost	Probable Vertex Cover	Probable Cost
1	$\{0,1,2\}$	3	$\{0,1,2,3\}$	4
2	$\{0,1,2,3\}$	4	$\{0,1,2,3\}$	4
3	$\{0,2,3\}$	3	$\{0,1,2,3\}$	4
4	$\{0,1,2,3\}$	4	$\{0,1,2,3,4,5,6,7\}$	8
5	$\{0,2,3\}$	3	$\{0,1,2,3\}$	4
6	$\{0,2,4,6\}$	4	$\{0,1,2,3,4,5,6,7\}$	8

6 Room for Improvement

For bipartite graphs, the result from approximation algorithm is almost twice for most of the cases. There is a room for improvement in case of bipartite graphs. Kőnig's theorem states that, in any bipartite graph, the minimum vertex cover set and the maximum matching set have in fact the same size. So if we are given a bipartite graph we can solve the maximum matching problem in that graph to determine the minimum vertex cover. Maximum matching can be done in $O\left(|V|\sqrt{|E|}\right)$ time using Hopcroft–Karp algorithm.

For general graphs, we can increase the number of iteration to get a better result.

7 Conclusion

Vertex cover optimization serves as a model for many real-world and theoretical problems. The problem has been used to model the elimination of repetitive DNA sequences for synthetic biology and metabolic engineering applications. But whether or not it can be solved in polynomial time is still a question of interest.