Algorithms for Minimum Vertex Cover: Theory and Experiments

A Project Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Bachelor of Technology

by

Suman Saurav Panda
(111501037)

under the guidance of

Dr. Krithika Ramaswamy

INDIAN INSTITUTE
OF TECHNOLOGY
PALAKKAD

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

*This is to certify that the work contained in this thesis entitled "**Algorithms for Minimum Vertex Cover: Theory and Experiments**" is a bonafide work of **Suman Saurav Panda** (**Roll No. 111501037**), carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Palakkad under my supervision and that it has not been submitted elsewhere for a degree.*

Dr. Krithika Ramaswamy

Assistant Professor

Department of Computer Science & Engineering

Indian Institute of Technology Palakkad

Acknowledgement

I am thankful to IIT Palakkad for providing me the opportunity to do this project. I extend my heartiest thanks to Dr. Krithika Ramaswamy for supporting me to achieve this goal. I would also like to acknowledge Dr. Jasine Babu for her help and suggestions in my work.

Abstract

The aim of the project is to study how efficient theoretical algorithms for the classical Vertex Cover *problem are in practice. First, we study reduction techniques, exact algorithms and heuristics to find a minimum vertex cover. Then, we implement these algorithms and study how well they perform on instances that may be useful in practice. Finally, we make certain observations on the results obtained and conclude with directions for future work.*

Contents

Chapter 1

Introduction

An undirected graph is a pair consisting of a set V of vertices and a set E of edges where $E \subseteq V \times V$. An edge is specified as an unordered pair of vertices. For a graph G, $V(G)$ and $E(G)$ denote the set of vertices and edges respectively.

A vertex cover of a graph is a subset of its vertices containing at least one endpoint of each of its edges. We say that a vertex cover covers the edges of the graph.

Definition (Vertex cover): A vertex cover V' of a graph G is a subset of $V(G)$ such that for every $uv \in E(G)$, either $u \in V'$ or $v \in V'$, i.e., V' is a set of vertices that contains at least one endpoint of every edge.

Clearly, for any graph G, $V(G)$ is a vertex cover and the Minimum Vertex Cover problem is the task of computing a vertex cover of minimum size.

Minimum Vertex Cover
Input: A graph G
Output: A minimum vertex cover of G

Minimum Vertex Cover is a classical graph-theoretic problem that has several applications in various areas [1]. The decision version of this problem is defined as follows.

VERTEX COVER

Input: A graph G and an integer k

Question: Does G have a vertex cover of size at most k?

A problem is said to be polynomial-time solvable (or in the class P) if it can be solved in time polynomial in the size of its input instance. A problem is said to be in the class NP (non-deterministic polynomial time) if a candidate solution can be verified whether it is indeed a solution or not in polynomial (in the size of the input) time. It is well-known that $\mathsf{P} \subseteq \mathsf{NP}$ and whether this inclusion is strict or not is a major open problem in Computer Science. A problem Q is said to be NP-hard if every problem in NP reduces in polynomial time to Q. Such a problem is said to be at least as hard as any problem in NP. In other words, solving Q in polynomial-time implies that $\mathsf{P} = \mathsf{NP}$. A problem is said to be NP-complete if it is NP-hard, and also in NP.

MINIMUM VERTEX COVER is a well-known NP-hard problem and its decision version referred to as VERTEX COVER is known to be NP-complete [2]. Due to its wide applicability in several areas, there have been several approaches described to solve MINIMUM VERTEX COVER (and its decision version) in the literature. In this work, we study such algorithms and analyze their performance on real-time instances.

Chapter 2

Review of Prior Works

All graph-theoretic terminology and notation not defined here can be found in [3]

As we have mentioned earlier in the introduction VERTEX COVER is NP-complete and we don't have any efficient algorithm to solve it in polynomial time. A trivial algorithm to find a MINIMUM VERTEX COVER is to enumerate all possible subsets of the vertex set and check if each such set is a VERTEX COVER. This brute force algorithm runs in $O(2^n n^2)$ time where n is the number of vertices in the input graph. However, this algorithm is infeasible in practice as even for $n = 100$, we need to try 2^{100} possibilities. This is clearly an inefficient algorithm.

One of the approaches to NP-hardness is the framework of parameterized algorithms. A parameterized problem is a language $Q \in \Sigma^* \times N$, where Σ is a fixed finite alphabet. An instance of Q is a pair $(x, k) \in \Sigma^* \times N$, where k is called the parameter. A parameterized problem Q is said to be fixed Parameter tractable (FPT) if there exists an algorithm $\mathcal{A}$, a computable function $f : N \to N$, and a constant c such that given $(x, k) \in \Sigma^* \times N$, $\mathcal{A}$ correctly decides whether $(x, k) \in Q$ or not in time upper-bounded by $f(k)\dot{|}(x, k)|^c$. The complexity class containing all fixed-parameter tractable problems is called FPT.

A kernelization algorithm for a parameterized problem Q is an algorithm $\mathcal{A}$ that, given an instance (x, k) of Q, works in polynomial time and returns an equivalent instance (x', k')

of Q with $|x'| + k' \leq g(k)$ for some computable function $g : N \to N$.

In this chapter, we list some of the well-known approaches to solve . All these results can be found in [3]. Before proceeding to these results, we list some simple rules that may reduce the size of an input instance.

Pendant edge rule: If G has an edge uv with deg(u) = 1, then reduce (G, k) to $(G - \{u, v\}, k - 1)$.

Domination rule: If G has two adjacent vertices u and v such that $N(v) \subseteq N[u]$, then reduce (G, k) to $(G - u, k - 1)$.

2.1 Buss Reduction

The first kernelization algorithm for VERTEX COVER, attributed to Buss, relies on two observations.

- No minimal contains an isolated vertex as it does not have any incident edges that need to be covered.
- If v is a vertex of degree more than k, then any of size at most k contains v. If we do not include v, we would have to include all of its at least $k + 1$ neighbors, exceeding the budget of k.

Isolated vertex rule: If G has an isolated vertex v, then reduce (G, k) to $(G - v, k)$.

High degree vertex rule: If G has a vertex v with more than k neighbours, then reduce (G, k) to $(G - v, k - 1)$.

Applying these rules leads to the following result.

Theorem: 1 VERTEX COVER *admits a kernel with at most* $k^2 + k$ *vertices and* k^2 *edges.*

2.2 Crown Reduction

Crown decomposition is a general kernelization technique that can be used to obtain kernels for many problems.

Definition (Crown decomposition): Let G be a graph. A crown decomposition of G is a partition (C, H, B) of $V(G)$ such that the following properties hold.

- C is a non-empty independent set in G.
- There are no edges between vertices in C and vertices in B.
- $G[C, H]$ contains a matching that saturates H. where $G[X, Y]$ is the bipartite sub graph of G induced by vertex subset $X, Y \subseteq V(G)$

The following is a well-known result that leads to a kernel.

Theorem: 2 *(Crown Theorem) Let G be a graph without isolated vertices and with at least $3k + 1$ vertices. There is a polynomial-time algorithm that either*

- *Finds a matching of size$k + 1$ in G or*
- *Finds a crown decomposition of G.*

Theorem: 3 *admits a kernel with at most 3k vertices.*

2.3 Linear Programming Based Reduction

Many problems can be expressed in the language of Integer Linear Programmings (ILP). In an Integer Linear Programming instance, we are given a set of integer-valued variables, a set of linear inequalities (called constraints) and a linear cost function. The goal is to find an (integer) assignment of the variables that satisfies all constraints, and minimizes or maximizes the value of the cost function.

In this section we convert the problem into solving an integer linear program and then use it to a kernel with at most $2k$ vertices.

We introduce $n = |V(G)|$ variables, one variable x_v for each vertex $v \in V(G)$. Setting variable x_v to 1 means that v is in the , while setting x_v to 0 means that v is not in the . To ensure that every edge is covered, we introduce constraints $x_u + x_v \geq 1$ for every edge $uv \in E(G)$. The size of the is given by $\Sigma_{v \in V(G)} x_v$. In the end, we obtain the following ILP formulation: minimize $\Sigma_{v \in V(G)} x_v$ subject to $x_u + x_v \geq 1$ for every $uv \in E(G)$, $0 \leq x_v \leq 1$ for every $v \in V(G)$ and $x_v \in Z$ for every $v \in V(G)$.

As we can observe from the above, ILP is at least as hard as finding a MINIMUM . Hence we don't expect a polynomial time algorithm to solve this problem. However, we can do certain relaxations in the problem in order to get our desired kernel. The main hardness of ILP is the integrality constraint and we relax that constraint leading to a LINEAR PROGRAMMING problem defined as follows: minimize $\Sigma_{v \in V(G)} x_v$ subject to $x_u + x_v \geq 1$ for every $uv \in E(G)$ and $0 \leq x_v \leq 1$ for every $v \in V(G)$. For graph G we call this relaxation LPVC(G).

LPVC(G) can be solved in polynomial time and an optimum solution of LPVC(G) may help us towards the goal of finding a MINIMUM VERTEX COVER. Suppose we have got one optimum solution $(x)_{v \in V(G)}$ for LPVC(G). Now we partition the vertices according to their values assigned in the optimum solution as mentioned below.

- $V_0 = \{v \in V(G) : x_v < 1/2\}$
- $V_{1/2} = \{v \in V(G) : x_v = 1/2\}$
- $V_1 = \{v \in V(G) : x_v > 1/2\}$

Theorem: 4 *(Nemhauser-Trotter theorem) There is a* MINIMUM VERTEX COVER *S of G such that $V_1 \subseteq S \subseteq V_1 \cup V_{1/2}$.*

LP reduction rule: Let $(x_v)_{v \in V(G)}$ be an optimum solution to LPVC(G) in a VERTEX COVER instance (G, k) and let V_0, V_1 and $V_{1/2}$ be defined as above If $\Sigma_{v \in V(G)} x_v > k$, then

conclude that we are dealing with a no-instance. Otherwise, add V_1 to the solution, delete all vertices of $V_0 \cup V_1$, and decrease k by $|V_1|$.

This leads to the following result.

Theorem: 5 Vertex Cover *admits a kernel with at most* $2k$ *vertices.*

Chapter 3

Different Algorithms for Finding a Minimum Vertex Cover

In this chapter, we describe the different algorithms implemented for finding a minimum vertex cover and analyze their performance on certain instances. These instances can be found in [4].

3.1 Description of the Algorithms

We note that solving MINIMUM VERTEX COVER is polynomially equivalent to solving VERTEX COVER due to the following observation. Let $\mathcal{M}$ be a maximal matching (maximal set independent edges) in the input graph G. Such a set can be obtained by a greedy strategy in polynomial time. As no two edges in $\mathcal{M}$ share any endpoint, the size of any VERTEX COVER is at least $|\mathcal{M}|$ as we have to add at least one vertex from each edge in $|\mathcal{M}|$ to cover all the edges in $|\mathcal{M}|$. Hence we get a lower bound on the size of a MINIMUM VERTEX COVER.

Similarly in order to get an upper bound we first observe that the set S consisting of all the endpoints of the edges in $\mathcal{M}$ is a vertex cover. This is because by definition, we cannot add any edge from $G - S$ to $\mathcal{M}$ and get a bigger matching. That is, any edge in

the graph is adjacent to at least one edge in $\mathcal{M}$. Equivalently, the vertices of $G - S$ form an independent set (a set of pairwise non-adjacent vertices). Hence the MINIMUM VERTEX COVER size is upper bounded by twice the size of a maximal mathcing in the graph.

Now as we have an upper bound and a lower bound on the size of a MINIMUM VERTEX COVER, we can solve the MINIMUM VERTEX COVER problem by solving the VERTEX COVER problem for values of k in $\{|\mathcal{M}|, |\mathcal{M}|+1, |\mathcal{M}|+2, \ldots, 2|\mathcal{M}|\}$ with only a polynomial overhead in the running time.

Subsequently, let n denote the number of vertices in the input graph G of the VERTEX COVER instance (G, k). We also assume that G has no isolated vertices.

Algorithm 1: The first algorithm we implement is the simple combinatorial approach where we consider all 2^n subsets of the vertex set and check if each one is a VERTEX COVER or not. If one such subset that is a VERTEX COVER is of size at most k, then we conclude that (G, k) is an yes-instance. If no such set exists, then we conclude that (G, k) is a no-instance. This algorithm runs in $O(2^n n^2)$ time.

Algorithm 2: The second algorithm is similar to the first except that instead of trying all possible subsets of the vertex set, we try only k sized subsets. This algorithm runs in $O(\binom{n}{k} n^2)$ time. For small values of k, this is an efficient algorithm. For example, for $k \leq 5$, this is a $O(n^7)$-time algorithm.

Algorithm 3: The third algorithm is an FPT algorithm. We take an arbitrary vertex v that has at least one neighbour and branch on finding a VERTEX COVER including v and finding a VERTEX COVER excluding v. That is, in one branch we recurse on $(G - v, k - 1)$ and in the other we recurse on $(G - N[v], k - |N(v)|)$. The second branch follows from the fact that any veretx cover excluding v needs to include all its neighbors in order to cover all the edges incident to v. If we carefully observe, we note that the maximum depth of the recursion tree is k as we are interested in finding an at most k sized VERTEX COVER. Hence this algorithm runs in $O(2^k n^2)$ time.

Algorithm 4: In the next algorithm, we implement different reduction techniques in order

to get a relatively smaller instance of VERTEX COVER and then use the above combinatorial algorithms to find the required VERTEX COVER. The reduction techniques used are the ones described in the previous chapter. We use all three reduction techniques followed by Algorithm 3. The order of the reduction techniques is Buss reduction, crown reduction and LP reduction.

Algorithm 5: Here, we implement a randomized version of Algorithm 3. We take an arbitrary vertex v that has at least one neighbour. We generate a random bit $b \in \{0, 1\}$. If $b = 0$, we branch on finding a vertex cover including v and if $b = 1$, we branch on finding a vertex cover excluding v.

Algorithm 6: In this algorithm, we use all three reduction techniques followed by Algorithm 5.

The source code can be found in github repo.

3.2 Experimental Results

The result obtained by running the above six algorithms is documented in Tables 3.1, 3.2 and 3.3.

3.2.1 Average Case Analysis

As expected, the normal brute force algorithm of trying all possible subsets of vertices (Algorithm 1) shows poor performance for these graphs. The combinatorial approach where we enumerate a particular sized subsets of vertices (Algorithm 2) outperforms the brute-force approach. From the results, we observe that the algorithms which use reduction techniques followed by branching algorithm do not perform as well as expected for small graphs. Even Algorithm 2 performs better than both Algorithm 4 and Algorithm 6 in practice for smaller graphs. When we compare Algorithm 4 and Algorithm 6, for most of the graphs, Algorithm 6 is better but with a small margin, i.e., randomized branching computes the solution faster than the normal branching algorithm. Even though theoretically both these

Algorithm/Instances	A1	A2	A3	A4	A5	A6
I1-I50	145036	77946	6066	105495	5865	104214
I51-I100	104071	51185	5186	120219	5285	112934
I101-I150	94691	49991	5081	90143	5507	88698
I151-I200	161559	79220	6029	111803	6835	106238
I201-I250	106796	52390	5032	90556	5068	89298
I251-I300	117343	51788	5334	102426	5677	102145
I301-I350	150456	61003	5648	122176	6252	123038
I351-I400	104590	48578	5610	117365	5677	116378
I401-I450	98950	45527	4897	114752	6337	112597
I451-I500	152335	66103	5573	142316	6107	142165
I501-I550	113278	57457	5092	104098	5370	103428
I551-I600	142646	66452	6302	122788	6032	121684
I601-I650	159218	66123	6196	126362	5866	127869
I651-I700	149646	66096	5300	112311	5441	112665
I701-I750	159852	67385	6377	125529	6634	126252
I751-I800	88800	45223	5148	68140	5187	67447
I801-I850	171752	76010	5704	131210	6109	129698
I851-I900	187929	82591	5808	121160	6104	120832
I901-I950	120254	58615	5441	119897	5552	116108
I951-I1000	118816	56010	5604	92006	6388	92265
I1001-I1050	147987	54896	5261	119684	5743	116653
I1051-I1100	128373	53673	5732	112173	5868	113321
I1101-I1150	90564	53362	5239	89443	5566	88662
I1151-I1200	123886	48577	4907	119470	7348	120033
I1201-I1250	175701	65358	5276	112507	5727	113692
I1251-I1300	113059	52512	5097	100548	5114	103147
I1301-I1350	250404	86048	5536	99627	6082	99064
I1351-I1400	203816	81770	5938	130391	6146	130380
I1401-I1450	154141	80080	5841	127374	5920	126485
I1451-I1500	119295	66441	5104	101332	5511	97755

Table 3.1 Comparision of the average running times (in μs) of the algorithms on the instances

Algorithm/Instances	A1	A2	A3	A4	A5	A6
I1-I50	801856	602792	21660	572501	14996	571765
I51-I100	$> 10^6$	447389	16576	503315	15355	409439
I101-I150	570149	273409	14576	386473	17539	398568
I151-I200	$> 10^6$	$> 10^6$	15092	458091	19994	457503
I201-I250	727129	343341	13731	427937	11936	417661
I251-I300	701023	308203	12001	479945	12800	470440
I301-I350	$> 10^6$	565950	14231	477682	17990	479470
I351-I400	399069	197777	12389	471789	12395	458785
I401-I450	747486	293197	10312	386948	37024	394262
I451-I500	853231	236634	11379	468215	12783	482208
I501-I550	652254	274065	15320	467936	16201	457183
I551-I600	623664	239274	13832	472006	12431	466253
I601-I650	$> 10^6$	375812	13447	387580	14707	385322
I651-I700	$> 10^6$	975899	15671	469338	15897	477845
I701-I750	$> 10^6$	335987	15565	476468	26945	473039
I751-I800	323177	151929	13248	299848	10998	300291
I801-I850	$> 10^6$	760111	16739	686595	17195	675470
I851-I900	$> 10^6$	$> 10^6$	13184	667573	15950	671374
I901-I950	993262	674696	20671	386567	21670	388560
I951-I1000	651874	273392	16214	400343	23035	428611
I1001-I1050	908574	214201	11901	473077	13506	468746
I1051-I1100	710847	275909	12939	481957	14947	476615
I1101-I1150	531794	299882	18075	389812	15530	387814
I1151-I1200	834042	202782	11428	464696	26742	476028
I1201-I1250	$> 10^6$	316141	14604	575179	14697	566289
I1251-I1300	764451	252879	12953	468605	11049	482795
I1301-I1350	$> 10^6$	398421	11109	474487	16668	476147
I1351-I1400	$> 10^6$	871653	14649	482272	16513	477629
I1401-I1450	$> 10^6$	658527	11624	469858	13475	460200
I1451-I1500	$> 10^6$	506174	11567	519111	12245	523992

Table 3.2 Comparision of the maximum running times (in μs) of the algorithms on the instances

Algorithm/Instances	A1	A2	A3	A4	A5	A6
I1-I50	1387	1320	866	571	823	861
I51-I100	2345	1866	1528	1381	1362	1414
I101-I150	2690	2398	1230	875	1474	1473
I151-I200	3799	3407	1479	830	2018	1087
I201-I250	1399	1314	1399	797	1075	1040
I251-I300	4571	2819	1382	1359	1826	1137
I301-I350	2223	2253	1084	790	1406	1069
I351-I400	1387	1561	1253	1040	1401	1322
I401-I450	1876	1977	1424	648	1555	992
I451-I500	1181	1914	979	907	837	1451
I501-I550	2309	2026	1063	799	1159	1039
I551-I600	4308	2916	1672	1205	1838	1461
I601-I650	2422	2900	1140	774	1342	1391
I651-I700	4365	3769	1781	1689	1981	2243
I701-I750	4497	6086	2154	1271	2000	1459
I751-I800	678	1167	680	651	856	910
I801-I850	1405	2150	1416	858	1558	1190
I851-I900	2398	2107	1252	652	1490	814
I901-I950	2664	2752	1691	977	1611	1398
I951-I1000	4400	2894	1306	1410	2343	1735
I1001-I1050	3667	3325	1736	1244	1935	1555
I1051-I1100	3885	2617	1380	1110	1384	1048
I1101-I1150	3046	2652	1382	855	2027	1128
I1151-I1200	3060	3948	2293	820	1389	1411
I1201-I1250	3524	2777	1303	811	1417	1104
I1251-I1300	1431	1970	993	946	927	1429
I1301-I1350	2302	1670	1109	726	1175	1339
I1351-I1400	3320	2709	1296	546	1258	685
I1401-I1450	2115	2838	1401	814	1410	1398
I1451-I1500	7257	3592	1742	1133	1795	1652

Table 3.3 Comparision of the minimum running times (in μs) of the algorithms on the instances

algorithms have asymptotically lesser running times than the normal branching algorithms (Algorithm 3 and Algorithm 5), for smaller instances, the naive branching is faster. The reduction of graph and the reconstruction overhead while applying the reduction techniques are the cause of the poor performance on the considered data set. However, for larger graphs where the exponential factor dominates the the polynomial factor, we expect that Algorithm 4 and Algorithm 6 will work better. In that case, we may give preference to Algorithm 6 in practice. Summarizing, we note that both Algorithm 3 and Algorithm 5 run significantly faster for small graphs of vertex size < 30 and edge set size < 40 in the considered data set. Moreover, Algorithm 3 performs better than Algorithm 5 in most of these cases.

3.2.2 Worst Case Analysis

Following are some key observations from the worst case analysis. Algorithm 1 and Algorithm 2 show really poor performance in the worst case scenario. For Algorithm1 this was also observed in the average case analysis but here Algorithm2 also performs bad in worst case scenario. Comparing naive branching algorithms with the algorithms using reduction techniques, we observe some interesting facts. In the worst case, Algorithm 4 and Algorithm 6 do not deviate much from the average running time but Algorithm 3 and Algorithm 5 deviate a lot. The difference in the worst case running times for Algorithm 3 and Algorithm 4 are significantly lesser than the average running time difference of the same. So we may conclude that Algorithm 4 would outperform Algorithm 3 on larger graphs. Again Algorithm 6 performs better than Algorithm 4, i.e., applying reduction techniques followed by randomized branching is better than applying reduction techniques followed by normal branching. In maximum running time analysis for Algorithm 3 and Algorithm 5, we are not able to arrive at any particular decision as each one outperforms the other in almost half of the instances. Overall, we may conclude that Algorithm 3 is the most efficient from this analysis.

3.2.3 Best Case Analysis

Here, the winner is Algorithm 4. It almost defeats all the other algorithms by a good margin and compute the solution faster. Also, Algorithm 6 outperforms Algorithm 3 in many cases. However, the randomized version is not very efficient as Algorithm 6 is worse than Algorithm 4 and Algorithm 5 is worse than Algorithm 3.

Chapter 4

Conclusion and Future Work

From Tables 3.1, 3.2 and 3.3 of the previous chapter, it is clear that Algorithms 3-6 (branching algorithm variants) perform better than Algorithms 1 and 2 (brute force variants). Among Algorithms 3-6, Algorithms 3 and 5 are simple branching algorithms while Algorithms 4 and 6 involve reasonably sophisticated reduction rules followed by branching. It is interesting and surprising to note that Algorithms 3 and 5 perform better than Algorithms 4 and 6. This is a counter-intuitive result as we would typically expect reduction rules to help reduce the size of the input instances considerably. Exploring the properties of real-time instances that make them amenable to effective reduction is an interesting direction of work. Also, improving the implemention of the described algorithms and developing heuristics that solve instances of the Parameterized Algorithms and Computational Experiments Challenge (PACE) is an inevitable future work.

References

[1] F. N. Abu-Khzam and R. L. Collins and M. R. Fellows and M. A. Langston and W. H. Suters and C. T.Symons, "Kernelization Algorithms for the Vertex Cover Problem: Theory and Experiments," in *Proceedings of the 6th Workshop on Algorithm Engineering and Experiments (ALENEX)*, 2004, pp. 62–69.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, 3rd ed. MIT Press, 2009.

[3] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh, *Parameterized Algorithms*, 1st ed. Springer, 2015.

[4] Y. K. James Cheng and W. Ng, "A graph synthetic generator," 2006.