Available online at www.sciencedirect.com
SciVerse ScienceDirect
Procedia Computer Science 9 (2012) 747 – 753
Procedia Computer Science

International Conference on Computational Science, ICCS 2012

A Simple NOVCA: Near Optimal Vertex Cover Algorithm

Sanjaya Gajurel*, Roger Bielefeld

Case Western Reserve University, Cleveland, OH 44106, US

Abstract

This paper describes an extremely fast polynomial time algorithm, the Near Optimal Vertex Cover Algorithm (NOVCA) that produces an optimal or near optimal vertex cover for any known undirected graph G (V, E). NOVCA constructs the vertex cover by repeatedly adding, at each step, all vertices adjacent to the vertex of minimal degree; in the case of a tie, it selects the one having the maximum sum of degrees of its neighbors. The results identifying bounds on the size of the minimum vertex cover as well as polynomial complexity of algorithm are given with experimental verification. Future research efforts will be directed at tuning the algorithm and providing proof for better approximation ratio with NOVCA compared to any other available vertex cover algorithms.

Keywords: Vertex Cover Problem, Combinatorial Problem, NP-Complete Problem, Approximation Algorithm

1. Introduction

The Vertex Cover (VC) of a graph G(V,E) with vertex set V and edge set E is a subset of vertices C of V ($C \subseteq V$) such that every edge of G has at least one endpoint in C. In 1972 Richard Karp [1] showed that identification of minimal VC in a graph is an NP-complete problem.

Vertex Cover has been actively studied because of its important research and application implications. Various algorithmic approaches have been used to tackle NP complete problems such as the VC problem. Polynomial-time approximation algorithms for VC have been developed but do not guarantee optimality. By using the definition of approximation ratio, VC has an approximation ratio of $\rho(n)$ for any input of size n. The solution C produced by approximation algorithm is within the factor of $\rho(n)$ of the solution C* of an optimal algorithm i.e. $C^*/C \leq \rho(n)$. Also, the approximation algorithm has approximation ratio of $2 - \varepsilon$, where $0 < \varepsilon < 1$. A 2-approximation [2] algorithm has been trivially obtained and similar approximation algorithms have been discovered [3] [4] with an achieved approximation of $(2 - (\ln(\ln n)/2\ln n))$, where n is the number of vertices. Halperin [5] achieved an approximation factor of $(2 - (1 - o(1))(2\ln(\ln \Delta)/\ln \Delta))$ with maximum degree at most Δ. Karakostas [6] achieved an approximation factor of $(2 - \theta(1/(\log n)^{1/2}))$, the best approximation yet, by using the semidefinite programming relaxation of VC. Evolutionary algorithms (EA) that are randomized search heuristics have also been used for

* Corresponding author. Tel.:1-216-368-5717
E-mail address: sxg125@case.edu

doi:10.1016/j.procs.2012.04.080

solving combinatorial optimization problems including VC [7] [8].

Vertex Cover problems have been solved in O (1.2738k + kn) time [9] by using a bounded search technique where a function of a parameter restricts the search space. Abu-Khazm et al. have identified crown structure to reduce the size of both n and k [10]. It has been known that when relevant parameters are fixed, NP-complete problems can be solved in polynomial time. In both [10] and [11], n is the input size and k is the positive integer parameter. Though not guaranteed to find a vertex cover, an approximation of 3/2 for almost every single graph was obtained in [11]. According to Dinur and Safra [12], it is NP-Hard to get $\varepsilon < 1.3606$.

The paper is organized as follows: the NOVCA algorithm is described in Section 2; Section 3 provides experimental results; Section 4 is the conclusion.

2. Near Optimal Vertex Cover Algorithm (NOVCA)

NOVCA is motivated by the fact that a vertex cover candidates are those that are adjacent to minimum degree vertex so that its degree will be forcibly rendered to zero without choosing it. This fact has been reinforced during tie when the vertex with neighbors having maximum degrees is preferred over other minimum vertices. Without any optimization effort, the complexity of NOVCA is O (V^2 log V); with V = n, the complexity becomes O (n^2 log n) which is polynomial. The pseudo-code of NOVCA is presented in Fig. 1. Network Bench Node Degree algorithm [13] has been applied to determine the degree of each node. Then, the sum of the degree of adjacent nodes for each node is calculated. Both these values are included as data structures in a node - deg[v]/adj_deg_sum[v] as showed in Fig. 2. Initially, vertex cover set VC is empty. The vertices are chosen in increasing order of their degrees i.e. the adjacent vertices of minimum degree vertex are included in VC first. The magic function GetMinVertex() breaks a tie in vertex degrees choosing the adjacent vertices of the selected minimum degree vertex having maximum adjacent sum of degrees. The idea is to forcibly render the low degree vertices to zero without choosing them.

```
Declarations:
    V is the set of vertices of G
    E is the set of edges of G
    deg[V] is an integer array indexed by V for a set
           of vertices V
    sum_adj_deg[V] is an integer array indexed by V for
                   a set of vertices V
    VC is the set of vertices comprising a vertex cover
    $Q_{sum\_adj\_deg}$ is the set of vertices having min deg[V]
           (local variable in GetMinVertex())

Functions:
    Degree(v) is the degree of the vertex v ϵ V
    Adj(v) gives the set of vertices that are adjacent
           to v ϵ V
    GetMinVertex() identifies the next adjacent
                   vertices to include in the cover

    Heap_MIN(deg) returns the value of min. deg[V]
    HEAP_MAX($Q_{sum\_adj\_deg}$) returns the vertex having max
                              $Q_{sum\_adj\_deg}$


   for each v ϵ V {
      deg[v] = Degree(v)
   }

   for each v ϵ V {

     sum_adj_deg[v]  =$\Sigma_{v' \epsilon Adj(v)} deg[v']$
   }
```

```
E′ = E
VC = ϕ

while (E′≠ ϕ){
  $v_c$ = GetMinVertex(deg, sum_adj_deg)
  VC = VC + { Adj($v_c$) }
  for each v ϵ Adj(Adj($v_c$)){
   E' = E – { (adj($v_c$), v) }
   deg[v] = deg[v] – 1
  }
  V = V – { Adj($v_c$) }
  for each v ϵ V{
        If (Adj(v) == ϕ) continue
        sum_adj_deg[v] = $\Sigma_{v' \epsilon Adj(v)} deg[v']$
  }
 } //end while

 /// Magic Function GetMinVertex() Declarations ///

 Vertex GetMinVertex(deg, sum_adj_deg){
   $Q_{sum\_adj\_deg}$ = ϕ
   vmin_deg = HEAP_MIN(deg)
   for each v ϵ V{
     If (deg[v] == vmin_deg)
       $Q_{sum\_adj\_deg}$ = $Q_{sum\_adj\_deg}$ + {v}
   }
   return Heap_MAX($Q_{sum\_adj\_deg}$)
 }
```

Fig. 1. Pseudo-code for NOVCA; E[G]: set of edges of graph G; VC: Vertex Cover Set; Q: Priority Queue

3. Experimental Work and Results

Simulations to corroborate the theoretical results have been conducted on the CWRU High Performance Computing Resource using compute nodes with 3.0 GHz Intel Xeon processors running Red Hat Enterprise Linux 4 and using the gcc 3.4.6 compiler. Simulations are performed in both serial and parallel environments.

Simulation results for all example graphs as described above always return optimal (minimum) vertex cover. We have selected Complete Graph as a test graph to determine time complexity of NOVCA for two reasons:

1. Optimal vertex cover is known; n – 1; where n is the number of vertices
2. requires exhaustive search; there is an edge from each vertex to all other vertices

The shell script in Fig. 2 “graph_gen.sh” generates a complete graph of size n entered as input. This graph is then fed to executable “vc (serial) or vc_openmp (parallel)” (C++ program compiled with g++ compiler) to get vertex cover for that particular graph. The outputs are showed in Fig. 3.

```
#PBS -l walltime=36:00:00
#PBS -l nodes=1:ppn=4:quad
#PBS -N graph1000
#PBS -j oe
cd $PBS_O_WORKDIR
/usr/local/bin/pbsdcp -s vc graph_gen.sh $TMPDIR
cd $TMPDIR
sh graph_gen.sh 1000
cp gen_graph graph1000
time ./vc graph1000 #vc_openmp for parallel
```

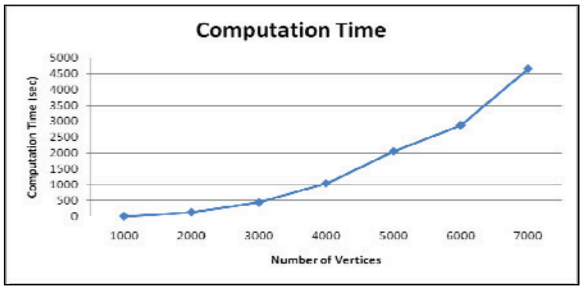
```
/usr/local/bin/pbsdcp -g '*' $PBS_O_WORKDIR
cd $PBS_O_WORKDIR
```

Fig. 2. The graph_gen.sh takes 1000 (number of vertices) as an input that creates a netlist in a file, graph1000, input to the executable vc; execuatable vc will be vc_openmp and ppn = 4 respectively for parallel implementation

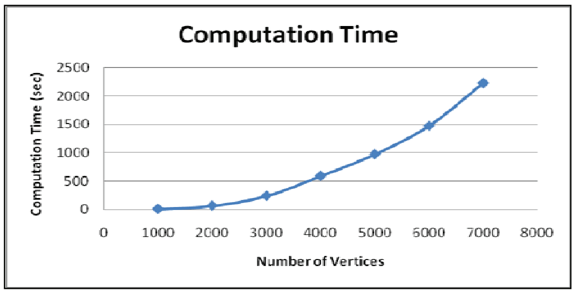
```
The cover consists of the following vertices:
  0      1      2      3      4      5      6      7
  8      9     10     11     12     13     14     15
…
…
994    995    996    997    998
There are 999 vertices in the cover.
real    0m7.161s
user    0m7.156s
sys     0m0.004s
```

Fig. 3. Output showing the vertices in a vertex cover, number of vertices, and execution time

We have recorded the computation time for different sizes of the graphs for both serial and parallel implementation to elucidate the polynomial complexity of NOVCA algorithm as depicted in Fig. 4(a)(b). We used MATLAB's polyfit(x,y,n) command to verify polynomiality as shown in Fig. 5 and Fig 6(a)(b).



(a)

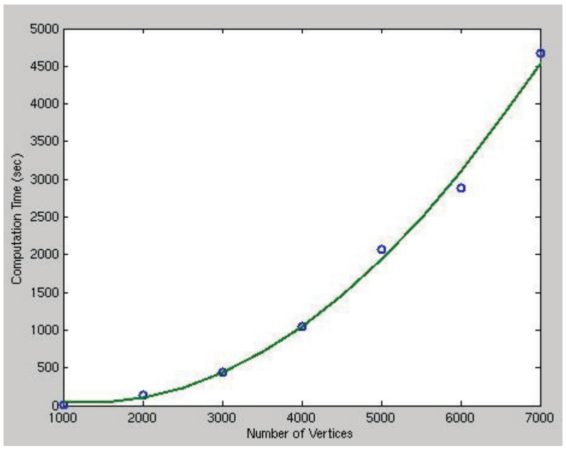


(b)

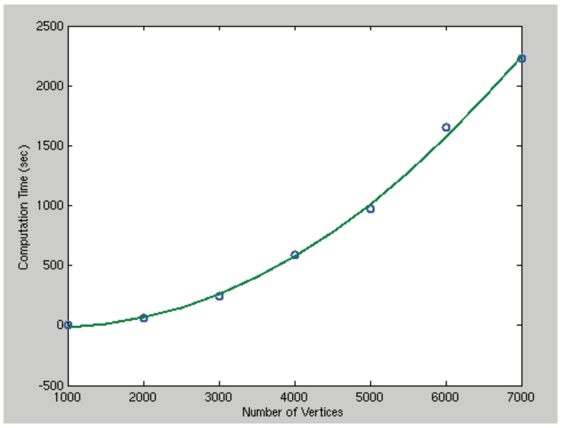
Fig. 4. Computational Time of NOVCA for different sizes of complete graphs for (a) Serial and (b) Parallel

```
x = [1000,2000,3000,4000,5000,6000,7000];
y=[7.124,129.21,437.274,1046.93,2061.037,2882.444,4666.
  976]; % from serial implementation
y=[7.083,65.08,238.669,589.784,971.582,1649.391,2223.02
  0]; % from parallel implementation
p = polyfit(x,y,2)
p = 0.0001   -0.3592  258.4364
x2 = 1000:500:7000;
y2 = polyval(p,x2);
plot(x,y,'o',x2,y2)
```

Fig. 5. MATLAB commands used for output data (computation time) from simulation for both serial and parallel implementation



(a)



(b)

Fig. 6. MATLAB plot using polyfit with n=2; (a) Serial and (b) Parallel

NOVCA has approximation ratio smaller than 1.3606 for all available bench mark (Table 1[14]; not showed all of the instances) graphs. For some instances like c-fat, Johnson, and random graphs NOVCA provides optimal cover. Noticeably, the execution time of NOVCA for any instance is remarkable. NOVCA has been found to perform very well compared to other available algorithms. For the instances where it provides near optimal solutions, it outperforms other algorithms in terms of execution time. We have compared NOVCA with COVER [15]. COVER is a stochastic local search algorithm for k-vertex cover. It constructs the initial candidate solution C greedily. When the several vertices satisfy the criterion for inclusion in C, COVER selects one of them randomly with uniform probabilities. The COVER algorithm terminates when either the vertex cover is found or max number of steps (MAX_ITERATIONS), has been reached. NOVCA, on the other hand doesn't have any randomness element and terminates when there are no more vertices in V. So, it has only one run unlike average execution time calculated using random seeds in different runs in COVER. Though COVER is found to obtain better vertex cover in most of the instances of the benchmarks, NOVCA is very simple and it outperforms COVER in execution time. In case of MANN_a81 where both NOVCA and COVER return the same value 2225, NOVCA is 20 times faster. Also, for the challenge instances of frb100-40 [14], NOVCA is off by just 17 vertices (NOVCA returns 3917 vertices whereas the optimal vertex cover is 3900), but the execution time is just remarkable; only 2013.667 sec. The challenge is stated as "Based on theoretical analysis and experimental results of smaller instances, I conjecture that in the next 20 years or more (from 2005), these two benchmarks cannot be solved on a PC (or alike) in a reasonable time (e.g. 1 day) [14]."

Table 1. NOVCA Performance Comparison between NOVCA and COVER on DIMACS and BHOSLIB benchmarks |V|: number of vertices; |C*|: optimal cover; NOVCA |C|: cover returned by NOVCA; COVER $|C|_{avg}$: Cover returned by COVER; NOVCA Time (sec): Execution time for NOVCA; COVER $Time_{avg}$: Average execution time for COVER

Instances	|V|	|C*|	NOVCA |C|	NOVCA |C|/|C*|	NOVCA Time (sec)	COVER $\|C\|_{avg}$	COVER $Time_{avg}$(sec)
frb59-26-1	1534	1475	1485	1.007	80.258	1477	18611.3
frb59-26-2	1534	1475	1484	1.006	79.297	1478	18589.5
frb100-40	4000	3900	3917	1.004	2013.667	-	-
broc200_1	200	179	181	1.011	0.115	179	768.2
broc800_4	800	774	782	1.010	10.832	775	4051.2
C2000.9	2000	1922	1932	1.005	207.060	1922	21489.7
c-fat200-5	200	142	142	1	0.092	142	1549.1
c-fat500-10	500	374	374	1	2.117	374	4401.2
gen200_p0.9_44	200	156	163	1.045	0.092	156	1543.6
hamming10-2	1024	512	512	1	10.297	512	2412.2
hamming10-4	1024	984	988	1.004	21.505	986	3457.6
johnson16-2-4	120	112	112	1	0.076	112	297.9
johnson32-2-4	496	480	480	1	2.273	480	2351.9
keller4	171	160	164	1.025	0.007	160	985.7
keller5	776	749	761	1.016	9.125	749	2364.9
MANN_a27	378	252	253	1.004	0.493	252	756.3
MANN_a81	3321	2221	2225	1.002	773.963	2225	15672.1
p_hat500-1	500	491	492	1.002	2.683	491	1810.2
p_hat1500-3	1500	1406	1414	1.006	74.991	450	1298.9
san200_0.7_1	200	170	183	1.077	0.117	170	713.7
san1000	1000	985	991	1.006	22.901	989	4972.8
sanr200_0.7	200	183	185	1.011	0.857	183	788.2
sanr400_0.7	400	379	382	1.008	1.030	380	2112.5
graph50-10	50	35	35	1	0.006	35	124.5
graph100-10	100	70	70	1	0.034	70	205.3
graph200-05	200	150	150	1	0.114	150	854.1
graph250-05	250	200	200	1	0.300	200	988.5
graph500-05	500	290	290	1	1.604	290	2255.2

4. Conclusion

NOVCA algorithm provides optimal or near optimal vertex cover for known benchmark graphs. The experimental results depict that the algorithm is extremely fast compared to other available algorithms such as COVER. In future, we will present mathematical proofs and show that approximation ratio is very close to 1 in general. Further research will be conducted to obtain the exact relationship for approximation ratio. The CWRU High Performance Computing resources will be considered for parallel computation for complex example graphs to reduce execution time.

Acknowledgement

I would like to thank Geeta Dahal and Pujan Joshi for suggesting counter examples to early versions of the algorithm.

References

1. R. Karp.: Reducibility among combinatorial problems. in R. E. Miller and J. W. Thatcher (eds.) Complexity of Computer Computations, Plenum Press, NY, pp. 85-103
2. T. Cormen, C. Leiserson, R. Rivest: Introduction to Algorithms, The MIT Press (2001)
3. R. Bar-Yehuda and S. Even: A local-ratio theorem for approximating the weighted vertex cover problem. Annals of Discrete Mathematics, 25, pp. 27-45 (1985)
4. B. Monien. and E. Speckenmeyer: Ramsey numbers and an approximation algorithm for the vertex cover problem. Acta Informatica, 22, pp. 115-123 (1985)
5. E. Halperin: Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. SIAM J. on Computing, 31(5), pp. 1608-1623 (2002). Also in Proc. of 11th SODA, pp. 329-337 (2000)
6. G. Karakostas: A better approximation ratio for the vertex cover problem, ICALP (2005)
7. G. Rudolph: Finite Markov chain results in evolutionary computation: A tour d'horizon, Fundamenta Informaticae, vol. 35, no. 1–4, pp. 67-89 (1998)
8. P. Oliveto, J. He, X. Yao: Evolutionary algorithms and the Vertex Cover problem, IEEE Congress (2007)
9. J. Chen, I. Kanj and G. Xia: Simplicity Is Beauty: Improved Upper Bounds for Vertex Cover. Technical report, Texas A&M University (2005)
10. F. Abu-Khazm, M. Fellows, M. Langston, and W. Suters: Crown Structures for Vertex Cover Kernelization, Theory Comput. Systems 41, 411--430 (2007)
11. E. Asgeirsson and C. Stein: Vertex Cover Approximation on Random Graphs", WEA 2007, LNCS 4525, pp. 285–296, 2007
12. I. Dinur and S. Safra. The importance of being biased. Technical Report TR01-104, ECCC, Dec. 2001.
13. Network Bench Node Degree, http://nwb.slis.indiana.edu/
14. Vertex Cover Benchmark Instances (DIMACS and BHOSLIB), http://www.cs.hbg.psu.edu/benchmarks/vertex_cover.html
15. S. Richter, M. Helmert, and C. Gretton: A Stochastic Local Search Approach to Vertex Cover, In Proceedings of the 30th German Conference of Artificial Intelligence (KI), 2007, pp 412-426.