A LOOK AT THE DIFFERENT APPROACHES TO SOLVE VERTEX COVER PROBLEM

Ramprasad Bommaganty, *School of Informatics and Computing, Indiana University, Bloomington, U.S.A*

***Abstract*—NP-complete problems are of high significance with respect to their practical implementation. There are many sets of NP-complete problems which are usually attempted to be solved using techniques such as approximation, randomization, parameterization and heuristics. One such NP-complete problem is the vertex-cover problem. In an undirected graph, the vertex-cover problem finds the minimum size of set of vertices such that all the edges of the graph are covered. This problem can be usually solved using the approximation technique of APPROX-VERTEX-COVER algorithm. Other such techniques include using the DFS algorithm in combination with APPROX-VERTEX-COVER algorithm, heuristics, greedy approaches etc. In this paper, we look at the different existing solutions to the vertex-cover problem and also formulate a new approach to solve it to get a close to optimal solution, in the best way possible.**

***Keywords*—Vertex Cover Problem, Articulation points, DFS, BFS.**

I. Introduction

A. P vs NP class of problems

Many of the sorting algorithms existing today for computational purposes are primarily based on the assumption that they are solvable in polynomial time. Algorithms like the not so Bubble Sort to the highly preferred Quick Sort are part of the problems that can be solved in polynomial time. However, in the highly complex field of computer science, there exist a certain set of problems that cannot be solved in polynomial time, or so it is stated. One prime example that is universally known is the famous Turings halting problem.

The problems that can actually be solved in polynomial time are known to belong to the P class. NP class problems are those which do not have adequate proof to classify them as having a polynomial running time or not a polynomial running time. NP class problems are those which can actually be solvable in polynomial time if a non-deterministic turing machine is used for computation. Interestingly, computer scientists have not been able to disprove the fact that NP problems might be having a polynomial time, which makes the ambiguity of P vs NP problems highly intriguing.

B. NP-Complete Problems

NP- Complete problems are a subclass of the NP class problems. NP- complete problems are those whose solutions can be verifiable in polynomial time. An interesting feature of NP-complete problems is that if one NP-Complete problem can be solvable in polynomial time, then every other NP - Complete problem can be solvable in polynomial time. This is because, one NP-Complete problem can be transformed into an another NP - Complete problem, thereby is one of them is solved in polynomial time, it gives an opportunity to solve the other one in polynomial time as well.

A list of some of the well known NP-Complete problems are :

1) Knapsack problem
2) Hamiltonian path problem
3) Travelling Salesman problem
4) Subset sum problem
5) Clique problem
6) Vertex Cover problem
7) Graph coloring problem

There are usually five techniques followed to get solve NP-Complete problems:

1) Approximation: An approximation algorithm is developed which gives a near to optimal solution than a complete optimal solution.
2) Randomization: Using random input sizes can lead to an approach which might fail with some small probability but gives a good estimate of an average case.
3) Restriction: Faster algorithms are possible by restricting the size of the input significantly. The crucial issue pertaining to NP - Complete problems is that even though they look solvable, a moderate input size also might take a really long time to be solved.
4) Parameterization: Using parameters might be one way of solving the NP-Complete problems.
5) Heuristics: A heuristic is a any approach which does not guarantee an optimal solution but which can be useful to find solutions to NP - Complete problems.

C. THE VERTEX COVER PROBLEM

A vertex cover of an undirected graph G = (V,E) is a subset V V such that if (u,v) E, then u V or v V (or both). The vertex cover problem is to find a vertex cover of minimum size in a given graph.[1]

D. BICONNECTED COMPONENTS AND ARTICULATION POINTS

A biconnected component is a maximal biconnected subgraph and a tree of biconnected components is called a block cut tree of the graph. The blocks of a graph are connected to each other at certain shared vertices known as the articulation points. Articulation points are those shared vertices

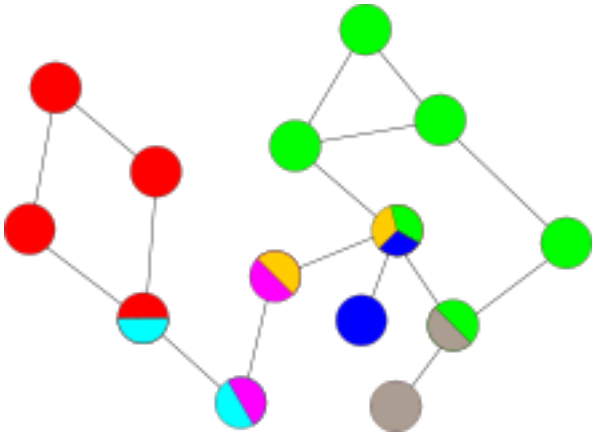


Fig. 1. The multi-colored nodes in the graph represent the articulation points which belong to multiple bi-connected components.

that increase the number of disconnections in the graph when removed.[2]

It is important to note that vertices which are part of multiple biconnected components can be identified as articulation points.

The figure above, taken from Wikipedia demonstrates the articulation points(or cut vertices as they are also known):

II. VARIOUS EXISTING APPROACHES

A. STANDARD APPROXIMATION APPROACH

The standard approach suggested in Cormen[1] is a polynomial time 2 approximation algorithm. If G(V,E) represents a graph consisting of V vertices and E edges, then the running time for this algorithm is O(V+E). The standard approximation algorithm is given as follows:

Algorithm 1 Algorithm - Standard Approximation for Vertex Cover problem

```
1: procedure APPROX VERTEX COVER(G)
2:     C ← ∅
3:     E' ← E[G]
4:     while E' ← ∅ do
5:         Let (u,v) be an arbitrary edge of E'
6:         C ← C ∪ {u,v}
7:         Remove from E' every edge incident on either u or v
8:     return C
```

B. NEARLY OPTIMAL VERTEX COVER (NOVAC -1)

This algorithm was proposed by Gajurel and Bielfeld in their paper A Simple NOVCA : Near Optimal Vertex Cover algorithm[8]. It is based on the greedy approach and works on the theory that highly probable candidates for the minimal vertex cover are connected to the minimum degree vertex. The degree of each vertex is calculated and the sum of the degree of the adjacent vertices is also calculated for each vertex of the graph. The vertex minimum degree is selected and all of its adjacent vertices are also added to the vertex cover. If a situation arises where two vertices have the same minimum degree then the sum of adjacent vertices of both of them is computed and the vertex whose sum of adjacent vertices is better is selected. Then its corresponding adjacent vertices are added to the vertex cover. This process repeats till all the edges in the graph are covered.

C. ADVANCED VERTEX SUPPORT ALGORITHM

The algorithm Vertex Support Algorithm proposed by Balaji, Swaminathan and Kannan[6] that defines a new feature for each vertex called its support. A support for any vertex of a graph is the sum of degrees of all its adjacent vertices. A vertex with maximum support is selected for the vertex cover and its corresponding edges are deleted.

Proposed by Khan, Ahmad and Khan in their paper AVSA, Modified Vertex Support Algorithm for Approximation of MVC[7], this algorithm is an improvement over Vertex Support Algorithm. It calcuates the support for each vertex in the graph, similar to VSA but instead chooses vertices with minimum support and adds it to a set. Then, among the nodes in the minimum support set each one is selected and the neighbour with maximum support value is added to the vertex cover set and its edges deleted. This process is repeated till all the edges have been processed.

D. GREEDY APPROACH BASED ALGORITHM

In their paper titled, A greedy approach based algorithm for the vertex cover problem Dimri, Purohit and Pant [8] proposed a greedy based approach to solve the vertex cover problem using adjacency matrices. The graph is represented in terms of an adjacency matrix with the initial matrix representing the connections between vertices. A symbol 1 signifies a presence of an edge between two vertices and 0 signifies no edge. The algorithm then iterates over all possible set of vertices, and the sum of the degrees of each row of the matrix is computed and denoted as D[i]. The maximum D[i] is denoted as Ki. The vertex corresponding to Ki is selected and added to the vertex cover. Its corresponding row and column values are changed to 0 in the adjacency matrix. Then, the next maximum D[i] is identified and this process repeats till all the values in the adjacency matrix are 0. The vertex set then obtained is the minimal vertex cover set for the graph.

The above three approaches mentioned are just few of the many proposed algorithms proposed for solving the vertex cover problem, but these seemed the most intuitive and practical approaches, at least for the graphs which were not huge in scale and followed a serial approach to solving the vertex cover problem.

In the next section, the basis for the proposed algorithm is introduced and the origin of the approach from a previous paper by Shah, Reddy and Selvakumar[5] is discussed.

III. PROPOSED APPROACH

A. Drawbacks of using DFS to find articulation points

According to the algorithm used by Shah, Reddy and Selvakumar in their paper Vertex Cover Problem - Revised

Approximation Algorithm [5], the procedure followed for finding the articulation points was to run a simple DFS search on the graph. This procedure used up a running time time of O(2(V+E)) which was asymptotically rounded off to O(V+E).

There is the other algorithm known as the Hopcroft-Tarjan algorithm that uses DFS to find the articulation points and is an optimal linear time sequential algorithm. A parallel algorithm known as the Tarjan-Vishkin parallel algorithm does not use DFS to find articulation points but uses an auxiliary graph to identify biconnected components in a graph.

However, the above methods are primarily sequential in nature in terms of execution, and even if they follow the parallel approach, they have not been thoroughly effective in certain cases.

B. USING THE PARALLEL BFS APPROACH TO IDENTIFY BiCC COMPONENTS

In their paper on finding strongly connected components Slota, Rajamanickam and Madduri developed a parallel BFS based approach to identify articulation points that is efficient when compared to running a simple DFS or the Hopcroft-Tarjan approach. According to their approach, a vertex can be identified as an articulation point if it has at least one child vertex that does not have a simple path in the graph to another vertex with the same BFS level as the former[3]. This algorithm works by running multiple Breadth first searches in parallel.

The algorithm is as follows:

Algorithm 2 BFS APPROACH TO IDENTIFY BiCC COMPONENTS

```
1:  procedure BFS-BICC(G(V,E))
2:      for all v ∈ V do Art(v) ← false
3:      Select a root vertex s
4:      Determine P and Levels from BFS(G,s)
5:      for all(u, v) ∈ E(V)andP[v] = u do
6:          ml ← BFS − ML(G(V \ {u}, E(V \ {u})), v)
7:          if ml − Levels[u] then
8:              Art(u) ← texttrue
9:              BREAK
10:     Check if s is an articulation point
```

In the above algorithm BFS-ML runs a new BFS search on the selected vertex u.

This algorithm is a fast way of processing all the articulation points in a graph because it is parallelized across all the selected vertices u of the graph. Also, the vertices which do have a child node are identified rather quickly since a vertex with a higher level is identified after about one or two frontier expansions in BFS-ML and the entire BFS need not be executed in BFS-ML [3]. Though this approach means that to identify if the root is an articulation point, one of its child nodes should reach all of the others, the paper claimed that in real world graphs there are many vertices with degree of one and identifying the neighbours of such vertices as articulation points mitigates the overhead involved in the aforementioned computation.

C. PROPOSED ALGORITHM

By running parallel BFSes on multiple systems, we aim to reduce the computation time that is needed to find all the articulation points. Then, we proceed to use the latter part of the algorithm proposed by Shah, Reddy and Selvakumar to identify an approximate vertex cover.

The algorithm is as follows:

Algorithm 3 Proposed Vertex Cover using BiCC

```
1:  procedure PROPOSED VERTEX COVER(G(V,E))
2:      C ← ∅
3:      C = BFS-BiCC(G(V,E))   ▷ all the articulation points
        are identified using the parallel approach of BFS-BiCC
4:      C ← C ∪ C'
5:      E' ← E[G]
6:      E' ← E'- {set of edges covered by vertices in C' }
7:      while E′ ≠ ∅ do
8:          if any edge (a,b) ∩ arbitrary edge of E' ≠ ∅then
9:              add either a or b to C and remove (a,b)
10:         else
12:             add the common vertex and
13:             remove edges from that vertex
15:
16:     return C
```

IV. ALGORITHM ANALYSIS

Since we have used a partly similar algorithm to find the minimal vertex cover as the one proposed by Shah, Reddy and Selvakumar, the number of elements in the vertex cover will be quite similar to the one obtained by them.

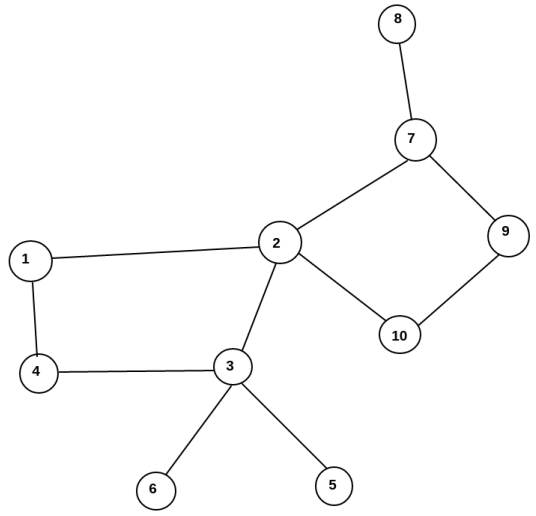
Using the similar analysis pattern as followed by them in their paper for a complex graph, but following our proposed algorithm.

This vertex cover contains 5 elements as opposed to the 8 elements we will get if the classical approach of finding the vertex cover as mentioned in Cormen[1] is followed. Our algorithm does not identify the vertex cover in a different way than the one proposed by Shah, Reddy and Selvakumar but where it hopes to get efficient is in terms of the running time needed to compute all the articulation points.

Using DFS to find articulation points takes O(2(V+E)) time but using BFS-BiCC algorithm to find the articulation points might take much less time because of the parallel approach to finding the articulation points, as opposed to the serial technique in DFS.

For instance, in their paper Slota, Rajamanickam and Madduri claimed that the parallel scaling of the BFS-BiCC has had an 8x speedup and good scaling across their three largest non-fully biconnected test graphs of Friendster, Orkut and Kron21. These results were in comparison to the serial approach followed to find the articulation points. While the exact running time of our algorithm is O(V+E), our hypothesis is that by running it on parallel systems for large graphs, it scales better and runs quicker than the O(2(V+E)) time needed by the serial DFS to find the articulation points. Graphs such as Friendster, Orkut and Kron21 are quite large and are representations of real time data with millions of edges and vertices. On

1. C <- ϕ
2. E' = {(1,2),(2,3),(3,4),(1,4),(2,7),(2,9),(2,10),(3,5),(3,6),(7,8),(7,9),(7,10),(9,10)}
3. Finding articulation points using BFS-BiCC, C' = {2,3,7}
4. C = {2,3,7}
5. E' = E' -{set of edges covered by vertices in C'}
6. E' = {(1,4),(9,10)}
7. Take (9,10) as an edge
8. C ={2,3,7,9}
9. E' = {(1,4)}
10. Take (1,4) as an edge
11. E' = { ϕ }
12. C = {1,2,3,7,9}



such graphs, our approach to parallelize the method to find the articulation points using the BFS-BiCC might work more efficiently. A test run on a huge network, preferably of any popular social network might deliver the appropriate results as mentioned by running on several distributed nodes.

V. CONCLUSION AND SCOPE FOR FUTURE IMPROVEMENTS

While the overall running time of the our proposed algorithm is O(V+E), the area where it aims to trump the classical approach for minimal vertex cover is by getting a smaller vertex cover set and also reduce the time taken to identify the articulation points.

For large networks where the practical implementations of vertex cover are huge, like for example, a social network or finding phylogenetic trees based on protein domain information[4] our algorithm can scale well and perform better than the one proposed by Shah, Reddy and Selvakumar, owing to the parallelization technique of implementing multiple BFSes.

Additional enhancements can be made to our algorithm by using a technique other than BFS-BiCC to identify articulation points or improving upon the greedy and advanced vertex cover identification techniques presented earlier. Using distributed systems of scale and big-data processing techniques to process large data is another improvement that can be worked upon for future improvements.

VI. ACKNOWLEDGEMENT

This project was done in association with the following students:

1) Achyut Sarma Boggaram(achbogga)
2) Harish Annavajjala(hannavaj)

I would also like to thank Prof Andy Somogyi, Indiana University - Bloomington.

VII. REFERENCES

[1] T.H. Cormen, C.H. Leiserson, R.L. Rivest and C.Stein ,Introduction to Algorithms - Third Edition, 2009 .

[2] Biconnected component, Wikipedia : https://en.wikipedia.org/wiki/Biconnected$_component$

[3] G.M.Slota, S.Rajamanickam and K.Madduri, BFS and Coloring-based Parallel Algorithms for Strongly Connected Components and Related Problems, The Pennsylvania State University, Sandia National Laboratories, 2014.

[4] F. Abu-Khzam, R. Collins, M. Fellows, M. Langston, W. Suters C. Symons, Kernelization Algorithms for the Vertex Cover Problem: Theory and Experiments, Proceedings of the 6th Workshop on Algorithm Engineering and Experiments (ALENEX), ACM/SIAM, Proc. Applied Mathematics 115, 2004.

[5] K.Shah, P. Reddy and R. Selvakumar, Vertex Cover Problem - Revised Approximation Algorithm, VIT University, 2015.

[6] S.Balaji, V.Swaminathan and K.Kannan, Optimization of Unweighted Minimum Vertex Cover, World Academy of Science, Engineering and technology, vol.67, 2010.

[7] I.Khan, I.Ahmad and M.Khan, AVSA, Modified Vertex Support Algorithm for Approximation of MVC, International Journal of Advanced Science and Technology, vol 67, 2014.

[8] S.C.Dimri, K.C.Purohit and D.Pant, A greedy approach based algorithm for the vertex cover problem, International Journal of Scientific Engineering Research Volume 4, Issue 3, 2013.

[9] I.Khan and S.Khan, Experimental Comparison of Five Approximation Algorithms for Minimum Vertex Cover, International Journal of U and E Service, Science and Technology, Vol.7, No.6, 2014.