3805ICT Advanced Algorithms

Assignment 2 (Q5)

Thien Phuc Vuong

S5193954

Thienphuc.vuong@griffithuni.edu.au

Table of Contents

[Introduction 3](#_Toc104903641)

[Algorithm Description 3](#_Toc104903642)

[Generated the complement graph. 3](#_Toc104903643)

[Maximized the intersection method 3](#_Toc104903644)

[Space Complexity 4](#_Toc104903645)

[Runtime Complexity 4](#_Toc104903646)

[Maximized the degree method 4](#_Toc104903647)

[Space Complexity 4](#_Toc104903648)

[Runtime Complexity 4](#_Toc104903649)

[Experimental Results 5](#_Toc104903650)

[Comparisons and Conclusion 6](#_Toc104903651)

[Literature Review 6](#_Toc104903652)

[References 6](#_Toc104903653)

# Introduction

This is the report of the question 5 from assignment 2 of Advance Algorithm course from Griffith University by Chris Vuong. The report states the solution for minimum vertex cover for the complement graphs of the supplied graphs. The author proposes two approaches: greedy intersection method and greedy degree method. Both approaches are not given the best minimum vertex cover, but it has approached some of the answers for the approximation of the minimum vertex cover.

After the algorithms are introduce, a literature review for comparing the difference approaches of minimum vertex cover.

# Algorithm Description

The data structure of the graph is using an adjacent list. If there is a need of using edges, a set of pairs of integers is created (for the first algorithm).

There are two approaches for the solutions, such as the maximized the intersection method and the greedy method.

## Generated the complement graph.

A text file documentation is used to produce the data input. The file's structure is divided into numerous lines, each of which contains the three main components. The first portion is the graph's character, although there are only two characters that need to be checked. They are 'e' for edges and 'p' for storing the graph's information (number of edges and number of vertexes). The second portion is the first vertex number and the last one is the second vertex number. Because 'p' always comes before 'e,' it can assign a full graph with the specified number of V vertices. The graph will then delete that vertex for each edge line.

## Maximized the intersection method

Maximized the intersection method pseudo-code

1. C = empty
2. E = Edges of a graph
3. While E is not empty:
   1. Let v be a vertex that intersects the most with E
   2. Put v in C
   3. Remove all edges relate to v on E
4. Return C

The intersection of the set Edges is used in this greedy strategy. The technique continuously reduces the set until it becomes empty by maximizing the vertex that appears in the set of E. This technique will always update the set edges. It removes the situation when a vertex has a high degree yet most of the edges have been utilized.

Functions for the algorithm:

* getMaxIntersection: This function is looping from 1 to V and finds the maximum intersection of the vertex. O(V2)
* countIntersection : This function is looping from all the edges of the vertex and if the edge is still in the set edges, it counts by 1. O(V)

### Space Complexity

The space for the graph is O(V\*E) and the space for the set of the edges is O( E)

### Runtime Complexity

The algorithm's most costly operation is 3a, in which the algorithm counts the maximum intersection each time before removing the vertex's edges. In the worst-case scenario, the loop might last up to V time. As a result, the algorithm has a runtime of O(V3)

## Maximized the degree method

This approach enhances the greedy algorithm above by keeping all vector degrees in an array and looping until all degrees are 0. Each degree is shown as an edge. The algorithms is tend to perform a quicker approach but still give the similar idea of the maximize the intersection algorithm.

Maximized the degree method pseudo-code

1. C = empty
2. Degree[V]= 0
3. For i<-0 to V:
   1. Degree[i] = graph[i].size()
4. Max = max(Degree)
5. While max != 0:
   1. Put max in C
   2. For v in graph[max]:
      1. Degree[v] -= 1
   3. Max = max(Degree)
6. Return C

In this algorithm, we assign the Van Emde Boas in question 3 for experiment whether it is applicable to optimize the runtime solution for this algorithm.

### Space Complexity

The space for the graph is O(V\*E).

### Runtime Complexity

The algorithm is the loop with maximum runtime is O(V). Each time it searches for the maximum degree, it needs to loop in V time. Therefore, the algorithm costs a run time of O(V2).

# Experimental Results

This is the experiment for the generation graph and the result generation between two algorithms. Moreover, another result in the second algorithm is made for comparing the use of Van Emde Boas Tree.

This table shows the result of MVC of each algorithm and its CPU times.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| File name | MVC was found | Maximized the intersection | | Maximized the degree (set) | | Maximized the degree (van tree) | |
| MVC | Time(sec) | MVC | Time(sec) | MVC | Time(sec) |
| brock800\_1 | 777 | 786 | 60 | 786 | 0.009 | 786 | 0.047 |
| brock800\_1 | 776 | 788 | 64 | 788 | 0.008 | 788 | 0.069 |
| brock800\_1 | 775 | 786 | 60 | 786 | 0.007 | 786 | 0.068 |
| brock800\_1 | 774 | 787 | 59 | 787 | 0.010 | 787 | 0.069 |
| c2000.9 | 1922 | 1944 | 386 | 1994 | 0.030 | 1944 | 0.550 |
| c4000.5 | 3982 | 3988 | 48347 | 3988 | 0.271 | N/A | N/A |
| MANN\_a45 | 691 | 705 | 0.400 | 705 | 0.002 | 705 | 0.003 |
| p\_hat1500-1 | 1488 | 1491 | 1042 | 1491 | 0.063 | 1491 | 0.302 |

This table shows the result of complement graph generation’s CPU times for each algorithm

|  |  |  |  |
| --- | --- | --- | --- |
| File name | Maximized the intersection (ms) | Maximized the degree (set) (ms) | Maximized the degree (van tree) (ms) |
| brock800\_1 | 2028 | 1126 | 2264 |
| brock800\_1 | 1777 | 1113 | 2121 |
| brock800\_1 | 1764 | 1093 | 2236 |
| brock800\_1 | 1792 | 1110 | 2190 |
| c2000.9 | 19126 | 9870 | 13774 |
| c4000.5 | 52425 | 27372 | N/A |
| MANN\_a45 | 3827 | 2753 | 5373 |
| p\_hat1500-1 | 4563 | 2194 | 6117 |

Difference of the greedy algorithms to the MVC:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| File name | MVC | Greedy algorithm MVC | Difference | Error different = Difference \*100 / MVC |
| brock800\_1 | 777 | 786 | 9 | 1.15 |
| brock800\_1 | 776 | 788 | 12 | 1.55 |
| brock800\_1 | 775 | 786 | 11 | 1.42 |
| brock800\_1 | 774 | 787 | 13 | 1.68 |
| c2000.9 | 1922 | 1944 | 22 | 1.14 |
| c4000.5 | 3982 | 3988 | 6 | 0.16 |
| MANN\_a45 | 691 | 705 | 14 | 2.02 |
| p\_hat1500-1 | 1488 | 1491 | 3 | 0.02 |
| Average | | | | 1.14 |

# Comparisons and Conclusion

Both methods provide the same number of results, as seen in the first table (different pattern in the code). The greedy method, on the other hand, is around 10000 times quicker than the intersection technique. The maximum the intersection technique, as shown in the algorithm description, uses more loops and has a runtime of O(V3), but the maximized degree approach has a runtime of just O(V2). When comparing the usage of set and van tree, the van tree is not constructed as a library, requires time to initialize, and has a restricted number of methods that may be called. Before it was broken, the van tree's restriction was accessible for about 3000 allocations.

For the graph generation, although the first algorithm takes a lot of time and space to store the graph, the van tree is still heavier for initializing and deployment.

In conclusion, the Greedy algorithm that applies the maximum degree is the quicker algorithm to find the vertex cover. It has an average of 1.14% higher than the minimum vertex cover that can be found in the other methods.

# Literature Review

In the research of Cai et al.(2013), It applied the local search to make the greedy search for the MVC. The algorithms assign each edge weight and when the MVC picks a vertex, the weight is continually updated. The NuMVC is at first may pick the vertex randomly but with more training, it finds the local minimum of the vertex cover. The research also compares its result to other heuristics to prove the effectiveness of the forgetting mechanism and the two-stage exchanges.

Another solution to solve minimum vertex cover is an approximation algorithm. In the research of Chen et al.(2016), he has proved that using the Dijkstra algorithm can solve MVC with the runtime of O(n4) with n is the number of vertices.

# References

Cai, S., Su, K., Luo, C., & Sattar, A. (2013). NuMVC: An Efficient Local Search Algorithm for Minimum Vertex Cover. Journal of Artificial Intelligence Research, 46, 687–716. <https://doi.org/10.1613/jair.3907>

Chen, J., Kou, L., & Cui, X. (2016). An Approximation Algorithm for the Minimum Vertex Cover Problem. Procedia Engineering, 137, 180–185. https://doi.org/10.1016/j.proeng.2016.01.248