ECE 650: Final Project

Efficiency Evaluation of Vertex Cover Optimization Approaches

Authors:
Aditya Subramanian, Masters in ECE
Safiya Jan, Masters in ECE

A report presented for:
Professor Reza Babaee

4/12/2020
University of Waterloo
Department of Electrical and Computer Engineering

Abstract

The Minimum Vertex Cover is a classic optimization problem defined as NP-Complete, meaning that it cannot be solved in polynomial time but rather, a solution to the instance of the problem can be verified in polynomial time. This report investigates the performance of three distinct Vertex Cover algorithms, namely: CNF-SAT-VC, Approx-VC-1 and Approx-VC-2, and compares the consequences of each through the computation of two efficiency metrics: ***run-time*** and ***approximation-ratio***. These algorithms are implemented in C++ and their corresponding efficiency metrics are derived through a sum total of 2100 runs through a graphical size rage of $[2, 50]$ using randomized graph generations produced by a graph generator, **graphGen**. The mean values for both efficiency metrics are calculated along with the standard variance and plotted as line graphs. Upon analysis, it is evident that CNF-SAT-VC produces the most accurate solution, but requires the largest amount of time. Approx-VC-2 produces the quickest solution in general, but compromises on accuracy. Approx-VC-1 is concluded to be the most feasible approach in terms of accuracy and time efficiency, as this approach produces a nearly optimal solution analogous to the CNF-SAT-VC approach, yet produces this solution almost 120 times quicker.

1.0 Introduction

The Vertex Cover problem is defined in the realm of discrete mathematics within a discipline known as graph theory. The definition is as such: the Vertex Cover of an undirected graph, $G = (V, E)$, is a subset of vertices, V', such that every edge in G has at least one endpoint in the Vertex Cover, V' [1]. In other words, the Vertex Cover is a subset of the vertices that completely cover all the edges in a given graph. The Vertex Cover problem has been a central problem in the study of parameterized algorithms and has applications in areas such as wireless networking, computer science and computational biochemistry [2]. It is often necessary to find the minimum Vertex Cover for a graph in order to optimize a solution to a given problem - such a task is categorized as an NP-Complete optimization problem. An NP-Complete problem is one that cannot be solved in polynomial time, but a solution however, can be verified in polynomial time [3].

This report analyzes the variation of efficiency for three algorithms in producing the minimum Vertex Cover using graphs of different sizes. Two metrics, namely ***run-time*** and ***approximation-ratio*** (ie. accuracy), are the main quantifiers used to evaluate algorithmic efficiency in this study and are discussed further in **Section 3.1: Efficiency Metrics** below.

2.0 Background Information

2.1 Optimal Approach: Reduction from Vertex Cover to CNF-SAT-VC

A Boolean Satisfiability Solver (ie. SAT Solver) is a tool which can be used to verify a solution to an instance of an NP-complete problem. The SAT solver harnesses algorithms that in some finite amount of computation, decide whether a given propositional logic formula is satisfiable [4]. The satisfying assignment from the SAT solver is then re-constructed to find the solution to a particular instance of the NP-complete problem under examination.

In the context of the Vertex Cover problem, a polynomial time reduction is performed on one graphical instance, *G*, and transformed into a propositional logic formula, *F*, in Conjunctive Normal Form (CNF). If the formula is satisfiable, the minimum vertex cover can be re-constructed from the satisfiability assignment. MiniSAT is used for this experiment due to its degree of modifiability and accessibility [5].

2.2 Approximation Approaches: Approx-VC-1 and Approx-VC-2

The approximation algorithms studied in this experiment, Approx-VC-1 and Approx-VC-2, are used to approximate solutions for instances of the Vertex Cover problem within polynomial time. However, as evident by the word "approximation", they only provide estimated solutions to each instance and therefore do not account for all edge cases which vary according to graph complexity [6].

In the Approx-VC-1 algorithm, the vertex of highest degree is added to the Vertex Cover, all edges incident to that vertex are discarded, and the process is repeated until no edges remain. In the Approx-VC-2 algorithm, an edge ⟨i,j⟩ is picked, both *i* and *j* are added to the Vertex Cover, all edges incident to both *i* and *j* are discarded, and the process is repeated until no edges remain.

3.0 Experimental Design

3.1 Efficiency Metrics

In order to evaluate the performance of each algorithm, two efficiency metrics are characterized: ***approximation ratio*** and ***run-time***. The approximation ratio is the ratio of the Vertex Cover produced by an approximation algorithm over the Vertex Cover produced by the optimal algorithm (CNF-SAT-VC), and is shown in **Equation 1** below:

$$AR = \frac{VC_{approximate}}{VC_{optimal}} \tag{1}$$

where:

AR = Approximation Ratio (1 is considered the most optimal)
$VC_{approximate}$ = Vertex Cover size produced by an approximation algorithm
$VC_{optimal}$ = Vertex Cover size produced by the optimal algorithm (ie. CNF-SAT-VC)

As mentioned in the project handout, the CNF-SAT-VC approach is guaranteed to be the optimal solution in producing the minimum Vertex Cover for a given graph [7], and is therefore the benchmark of correctness by which the approximation ratio seeks to evaluate all other algorithms with. The run-time is merely the amount of time each algorithm requires to produce a solution.

3.2 Methodology

In order to evaluate each algorithm according to the efficiency metrics defined above, a controlled testing scheme is implemented as follows:

Multi-threading is implemented to allow all algorithms to run concurrently, thereby enabling run-time tracking for each algorithm. When utilized correctly, multi-threading assists the operating system in allocating processor time more uniformly among different threads, thereby improving the applications processing time [8]. Due to the inability of the current encoding of the CNF-SAT approach to scale to large graphs, a timeout of two minutes is implemented into the CNF-SAT thread - the rationale for choosing this particular timeout is discussed further in **Section 4.1: Algorithm Run-Time Analysis**.

The algorithms are tested using random graphs generated by **graphGen** for a variety of graph sizes. Data collection is performed in the following way: the algorithms are used to generate Vertex Covers for ten graphs, run ten times, for 21 increments in the graphical size spectrum, $V = [2, 50]$, resulting in a sum total of 2100 runs. The increments used for the algorithms are as follows: increments of 1 from $[2, 15]$, and increments of 5 from $[20, 50]$. For each run, the run-time and AR are derived. Upon completing 100 runs for a size increment, the efficiency metrics are averaged and the standard deviation is calculated. The results are then plotted and analyzed, as shown in **Section 4: Results and Analysis**.

3.3 Relevant Implementation Details

A graph and its corresponding edge list is stored in the C++ program as an adjacency list - an array of vectors. The adjacency list representation provides a simple structure for the algorithms to compute their respective Vertex Covers. Furthermore, Approx-VC-2 is implemented to use the first edge in the adjacency list. These are important implementation details as they directly affect the run-time of both approximation algorithms, as discussed further in **Section 4.1: Algorithm Run-Time Analysis**. Lastly, the clauses provided in Assignment 4 are used for the encoding of the SAT solver and no further modifications have been made.

4.0 Results and Analysis

4.1 Algorithm Run-Time Analysis

The mean run-times for all algorithms are calculated and depicted in **Figures 1 and 2** below:

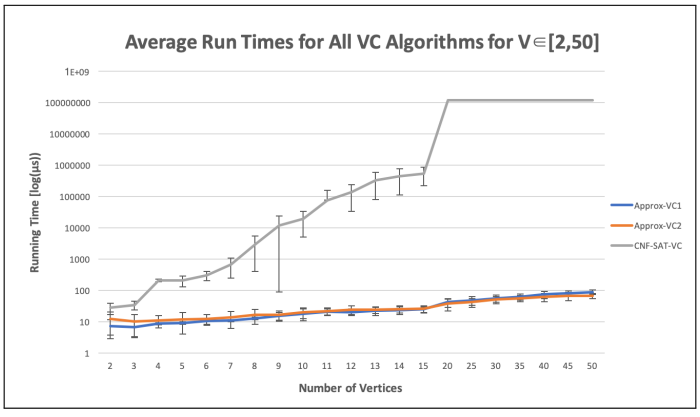

Figure 1: Mean run-times for all algorithms for all graph sizes

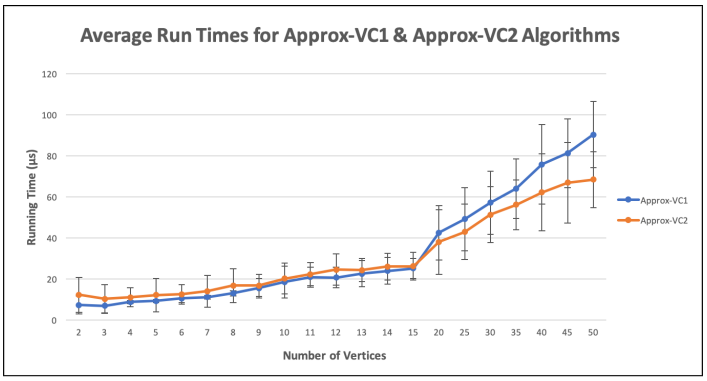

Figure 2: Mean run-times for approximation algorithms for all graph sizes

By examining the run-time plot in **Figure 1**, it is evident that CNF-SAT-VC exhibits the highest run-time among all algorithms due to its rapidly increasing slope. The run-time for CNF-SAT-VC grows exponentially as V increases. This is because as the graph size increases, the propositional logic formula, F, produced by the reduction grows rapidly - resulting in a higher processing time for the SAT Solver to compute a satisfying assignment. Note the plateau of CNF-SAT-VC after $V = 15$ at $\sim$1e+8 μs. This portion signifies the timeout feature of 2 minutes implemented for the CNF-SAT thread. For large graph sizes, specifically any graph larger than $V = 15$, CNF-SAT-VC requires an immense amount of time to produce a solution. Allocating a timeout larger than two minutes would render the CNF-SAT approach infeasible for two reasons.

Firstly, **Figure 2** shows that both approximations produce a result in less than a second regardless of the graph size in the test spectrum. Furthermore, the accuracy of Approx-VC-1 is nearly perfect as shown in **Figure 3** of **Section 4.2: Approximation Ratio Analysis**. In contrast to this rapid approximation, an allotted time of 2 minutes is a generous allowance for an algorithm which produces a slightly more accurate solution. Any algorithm in comparison to Approx-VC-1 which requires more than 2 minutes to produce a satisfiable result would in general, be too heavy of a time sacrifice for it to be practical in most cases. However, if the situation allows a relaxed time constraint but demands the most accurate solution possible, then the timeout would need to be reconsidered. Secondly, a timeout larger than two minutes would be infeasible in terms of data collection. Considering that data collection occurs from a sum total of 2100 application runs, it would take far too long to collect the data required to analyze the efficiency of each algorithm if the time-out was larger than two minutes. This timeout guarantees the approximation algorithms to produce a solution, while also giving a fair amount of time for the CNF-SAT-VC approach as well.

Figure 2 shows that both approximation algorithms have near linear run-times and perform almost equally well for graph sizes under $V < 15$, with Approx-VC-1 having a slight edge. However, for graphs sized $V > 15$, the run-time gap between the two trends starts to diverge, with Approx-VC-2 producing a solution significantly quicker. The longer run-time for Approx-VC-1 can be attributed to the implementation of the adjacency list, discussed in **Section 3.3: Relevant Implementation Details**. Approx-VC-1 must first find the highest incident vertex before every addition to its vertex cover, which requires the entire adjacency list

to be traversed each time. In contrast, Approx-VC2 uses the first available edge in the adjacency list, which eliminates the need for the entire adjacency list to be traversed. The positive correlation between graph size and adjacency list size thereby results in a longer processing time for Approx-VC-1. The presence of varying error bars in **Figures 1 and 2** are indicative of the correlation between processing time and machine configurability. The execution time varies for each run, resulting in the development of a standard deviation metric. Due to the possible degrees of graph complexity generated by **graphGen**, the time required for a vertex cover to be computed would vary among the algorithms, which further adds to the variation in run-times.

4.2 Approximation Ratio Analysis

The mean approximation ratios are calculated only in the size spectrum of [2,15] due to the inability of the CNF-SAT approach to produce a solution within the allotted time of 2 minutes for graphs bigger than $V = 15$, resulting in a total of 14 mean ratios. The results are shown in **Figure 3** below:

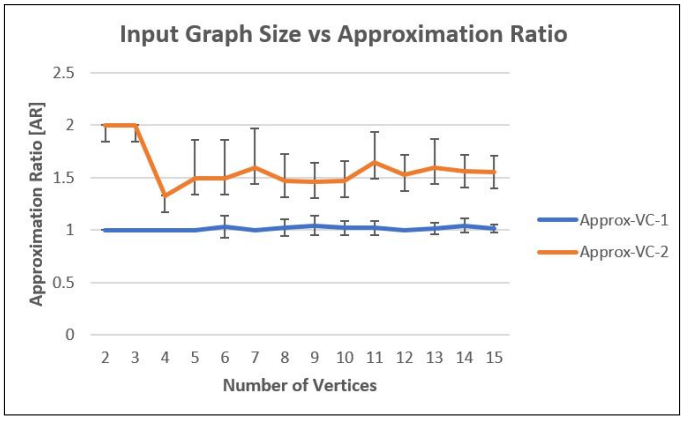

Figure 3: Input graph size vs approximation ratio

By examining the trend lines in **Figure 3**, it is evident that Approx-VC-1 is extremely effective in producing a nearly optimal solution whereas Approx-VC-2 produces a less accurate solution. The trend-line for Approx-VC-1 has a nearly horizontal profile converging closely with AR = 1, whereas the trend-line for Approx-VC-2 is more jagged and in general converges closer to AR = 1.5. The presence of error bars, which vary across both trend lines, can be attributed to the degree of graph complexity generated by **graphGen**. Since the algorithms generate approximate solutions to an NP-Complete problem, it is logical to conclude that the current implementation of these algorithms are not proficient enough to account for all edge cases which **graphGen** might produce. The trend line for Approx-VC-2 also consistently displays a significantly larger error bar for each graph size, which shows that Approx-VC-2 is more unstable in terms of repeatability. Lastly, the horizontal trend for size 2 and 3 converging at AR = 2 and the consequent downward spike in size 4 for the Approx-VC-2 trend-line can be attributed to the implementation details of the algorithm. For graphs sized 2 and 3, the minimum Vertex Cover will always be 1 as long as all vertices in the graph are connected. The implementation of Approx-VC-2 requires both vertices of an edge to be added to the Vertex Cover before deleting all edges incident to those vertices, thereby resulting in a Vertex Cover which is always an even number. Constraining the solution of Approx-VC-2 to produce

even-numbered results forces the algorithm to produce a solution which contains redundancies, thereby always resulting in a higher overall AR regardless of the graph size.

5.0 Conclusion

The conclusions derived from this experiment are as follows: although CNF-SAT-VC produces the most accurate result, its execution time exponentially increases for any increase in V, thereby making this approach infeasible for large graph sizes. Approx-VC-2 offers a solution to the Vertex Cover the quickest among all algorithms for graphs greater than $V = 15$, but achieves this speed due to a significant compromise in accuracy - it also always produces an even result for the vertex cover, which constrains the solution and advocates redundancies. Approx-VC-1 is the best approach in terms of accuracy and time-efficiency, as its AR closely aligns with 1 and the average execution time for all test runs is under one second. By comparing the two approximation algorithms, it can be concluded that even though Approx-VC-1 has a longer execution time than Approx-VC-2 for graphs sized $V > 15$, it produces a mean result which is $\sim$50 percent more accurate.

6.0 Limitations and Recommendations

One limitation of this experiment is the enormous amount of time required to truly compare the approximate algorithms to the optimal approach for the full range of graph sizes from [2,50] due to the inability of the CNF-SAT to scale for a large number of vertices. This can be addressed by improving the encoding of the SAT solver to extend its capability, and/or further dividing the CNF-SAT approach into multiple threads. Another limitation of this experiment is the presence of large error bars scattered amongst the plots. Data variance can be reduced by increasing the number of runs performed on each graph size, thereby resulting in more data points for the calculation of more refined mean values. Furthermore, a different graph generator could be substituted which guarantees a certain level of complexity among all randomized graph generations, which could further reduce the variance in run-time.

Bibliography

[1] Brilliant. *Vertex Cover*. 2018. URL: https://brilliant.org/wiki/vertex-cover/. (accessed: 04.13.2020).

[2] Jianer Chen et al. "Improved Parameterized Upper Bounds for Vertex Cover". In: *Mathematical Foundations of Computer Science 2006* 322.31 (2006), pp. 238–249. DOI: https://www.cs.lafayette.edu/~gexia/research/mfcs06.pdf.

[3] Reza Babaee. *Basics of Algorithms*. 2020. URL: https://ece.uwaterloo.ca/~rbabaeec/ece650/w20/assets/pdf/L6_algorithms.pdf.

[4] Reza Babaee. *SAT Solvers*. 2020. URL: https://ece.uwaterloo.ca/~rbabaeec/ece650/w20/assets/pdf/L10_sat.pdf.

[5] Niklas Sörensson Niklas Eén. *The MiniSat Page*. URL: http://minisat.se/.

[6] David Shmoys David P. Williamson. *The Design of Approximation Algorithms*. Cambridge University Press, 2011. ISBN: 9780521195270.

[7] Reza Babaee. "Final Course Project". In: *ECE 650: Methods and Tools for Software Engineering* (2020). DOI: https://ece.uwaterloo.ca/~rbabaeec/ece650/w20/.

[8] Evaluation Engineering. *The Advantages of Multithreaded Applications*. 1998. URL: https://www.evaluationengineering.com/home/article/13000965/the-advantages-of-multithreaded-applications.