Quality of Stochastic Versus Deterministic Algorithms for the Unweighted Minimum Vertex Cover Problem

Edmond Chuc

School of Information and Communications Technology
Griffith University
Gold Coast, QLD, 4215, Australia

14th September, 2018

Abstract

Given the rise of the World Wide Web and the accessibility of the internet, the importance of efficient and quality algorithms for solving the minimum vertex cover is greater than ever before. Such use cases for the minimum vertex cover problem can be placing security cameras around city blocks using a more optimal-bound stochastic algorithm while a fast deterministic algorithm can be applied to real-time applications. The stochastic local search algorithm and the deterministic algorithm have shown to produce quality results on the DIMACS benchmark graphs for solving the minimum vertex cover problem.

1 Introduction

In graph theory, a graph G=(V, E) contains a vertex cover V′ where it is a set of vertices (or nodes) which connects to every single edge in E of a graph G. The size of a vertex cover is the number of vertices within V′. The vertex cover was proven to be NP-complete by Karp's 21 NP-complete problems [4].

This proof garnered the interest of many researchers from around the world due to its applicability to real-world problems. Such world problems consist of race conditions in applications programming, optimising surveillance systems, communications systems and eliminating over-lap reads in bioinformatics. One particular research focus became the classical optimisation problem known as the minimum vertex cover (MVC).

The minimum vertex cover problem is to find the minimum number of vertices that connect to every single edge of a graph. In other words, it is to find the smallest V′ for a graph G. It is unlikely to find the optimal minimum vertex cover given that it is an NP-hard problem, however, many applications still exist for a solution that is *good enough* [3]. Due to this, many researchers have approached this problem with the use of approximation algorithms that give quality results in a reasonable time.

Parnas and Ron [5] approached the MVC by reducing a locally distributed approximation algorithm to a sublinear approximation algorithm. This was done by applying an *additive error* to the approximation algorithm. Pullan [6] extends the Phased Local Search (PLS), which was used for the maximum clique problem, for the MVC. PLS consists of three stochastic sub-algorithms as well as tabu search, which claimed state-of-the-art results in the year of its publication. Balaji et al. [1] designed the Vertex Support Algorithm (VSA) to find the small-

est vertex cover of a graph. The VSA keeps track of the graph by using an adjacency matrix and adds each vertex with the highest number of edges to the vertex cover. Ugurlu [8] designed a fast heuristic algorithm called Isolation Algorithm (IA), which isolates vertices with minimum degree edges and adds its adjacent neighbouring vertices to the vertex cover. Cai et al. [2] proposed NuMVC, an algorithm that matches state-of-the-art algorithms like PLS but in a more efficient manner. NuMVC uses two distinctive techniques, edge weighting with forgetting and two-stage exchange for efficiency. Tomar [7] proposed an improved greedy heuristic algorithm for the MVC problem which yielded good results for dense graphs. There were two versions of the greedy algorithm with different time complexities. One of them had a time complexity of $O(V \log V)$ while the other had $O(V^3)$.

Section 2 describes the proposed algorithms in this paper. Section 2.1 describes the stochastic local search algorithm. Section 2.2 describes the deterministic algorithm. In section 3, the results of the algorithms will be discussed and the conclusion can be found in section 4.

2 Proposed Algorithms

This paper proposes a stochastic local search algorithm for finding the minimum vertex cover. In each iteration, it randomly selects a vertex with the highest degree of neighbours and adds it to the vertex cover.

This stochastic local search algorithm (algorithm 1) and its results are compared to a naive, deterministic algorithm (algorithm 2). Algorithm 2 emphasises on speed while algorithm 1 focuses on lowering the vertex cover count and producing quality results. The deterministic algorithm run in $O(N)$ while the stochastic variant runs in $O(N * M)$ where M is the number of iterations to be performed.

2.1 Stochastic local search algorithm

Algorithm 1 operates as follows: The main algorithm body of the stochastic local search starts at line 8 and ends at line 27. Line 7 contains the `loop` which will run the main algorithm repeatedly until stopped.

Line 8 starts the first `loop` of the algorithm, which continues to run until all edges have been covered in the graph. In each iteration, it finds the highest degree number from the graph and then finds all the vertices with the degree equal to the highest as `verts`. A single vertex is randomly picked from `verts` and added to the vertex cover. The number of vertices with edges is then updated again before going to the next iteration. Once a vertex cover is found, the algorithm moves on to the second portion, starting at line 14. This line ensures that it keeps the lowest vertex cover in memory as it iterates over the outer `for loop`. Line 17 gets a list of vertices that are currently in the vertex cover as `vertices`. Line 18's `loop` ends when there are no more elements in `vertices`. The loop removes a vertex from the vertex cover and checks to see if it is still a valid cover. If it is not, it reverts back to the last known state while continuing forward with the current element to be removed from `vertices`. Otherwise if it is a valid vertex cover and its size is less than the current best vertex cover, then it is saved as the current best. The minimum vertex cover is found in `mvcBest`.

2.2 Deterministic algorithm

Algorithm 2 does some set up from lines 1-4 and the first `loop` of the algorithm starts on line 5 where it loops until there are no edges left in the graph. In each iteration, it finds the first vertex with the highest degree. It then adds it to the vertex cover and updates the graph by removing the edges that are covered by the chosen vertex. The second `loop` on starting on line 12 runs the same as algorithm 1's second `loop`. It removes each vertex and checks to see if it is still a valid vertex cover.

3 Results

Both the stochastic local search algorithm and the deterministic algorithm successfully obtained quality results, which can be found in table 1.

Algorithm 1: Stochastic Local Search

Result: Minimum Vertex Cover

```
1  graph ← readGraph(fileName);
2  graphOriginal ← graph;
3  mvc ← initMVC();
4  mvcOriginal ← mvc;
5  no ← vertsNoNeighbours(graph);
6  mvcBest ← mvc;
7  for i ← 0 to iterationMax by 1 do
8      while graph.size() is not no do
9          highestDegreeNo ← highestDegree(graph);
10         verts ← vertsWithDegree(graph, highestDegreeNo);
11         addVertexToMVC(graph, randomPick(verts), mvc);
12         no ← vertsNoNeighbours(graph);
13     end
14     if mvcBest.size() > mvc.size() or mvcBest.size() is 0 then
15         mvcBest ← mvc;
16     mvc ← mvcBest;
17     vertices ← mvc.getVertsAsVector();
18     while not vertices.isEmpty() do
19         mvc.remove(vertices[vertices.size()-1]);
20         vertices.pop();
21         valid ← validVC(graphOriginal, mvc);
22         if not valid then
23             mvc ← mvcBest;
24         else
25             if mvcBest.size() > mvc.size() then
26                 mvcBest ← mvc;
27     end
28 end
```

Algorithm 2: Greedy Local Search

```
Result: Minimum Vertex Cover
graph ← readGraph(fileName);
graphOriginal ← graph;
no ← vertsNoNeighbours(graph);
mvc ← initMVC();
while graph.size() is not no do
    V = pickVertex(graph);
    addVertToMVC(graph, V, mvc);
    no ← vertsNoNeighbours(graph);
end
mvcValid ← mvc;
vertices ← mvc.getVertsAsVector();
while not vertices.isEmpty() do
    mvc.remove(vertices[vertices.size()-1]);
    vertices.pop();
    valid ← validVC(graphOriginal, mvc);
    if not valid then
        mvc ← mvcValid;
    else
        if mvcValid.size() ¿ mvc.size() then
            mvcValid ← mvc;
end
```

Since the minimum vertex cover problem is NP-hard, the solutions found by the stochastic local search algorithm were considered good as they were all within 1% of the best minimum found from the DIMACS benchmark graphs. This is the same for the deterministic algorithm. The runtime comparison is considered to be equal since both algorithms performed similarly for a single run-through. However, it should be noted that the results of the stochastic local search were obtained only after running the algorithm N number of times. For example, this means that the algorithm took roughly 7-8 minutes for the testing machine to find the result for `brock800_4`, and since it's stochastic, there's no guarantee that it will find the same result in the same amount of time again. Due to its non-deterministic nature, sometimes the results were worse than the deterministic algorithm if the algorithm did not run for enough iterations. In comparison, the deterministic algorithm finds the result in `5ms` on average.

Due to the non-deterministic nature of the stochastic local search algorithm, it is suitable for applications which do not require real-time calculations. In contrast, though the deterministic algorithm performs marginally worse, it is well-suited for real-time applications for its speed and its deterministic nature.

4 Conclusion

The stochastic local search algorithm and the deterministic algorithm for the minimum vertex cover problem achieves quality results where both algorithms found results that are less than 1% off the minimum vertex cover of the DIMACS benchmark graphs. Although one performed marginally better than the other, the largest notable difference was in the speed in which the algorithms were required to perform. The stochastic local search algorithm found better results but requires more time, whereas the deterministic algorithm guarantees the same solution in a smaller amount of time. In conclusion, the trade-off is speed between the two algorithms. Both algorithms have their place in the world of software but their use depends on the requirements of the application.

References

[1] S. Balaji, Vishnu Swaminathan, and Krithivasan Kannan. Optimization of unweighted minimum vertex cover. 2012.

[2] Shaowei Cai, Kaile Su, Chuan Luo, and Abdul Sattar. Numvc: An efficient local search algorithm for minimum vertex cover. *CoRR*, abs/1402.0584, 2014.

[3] Irit Dinur and Samuel Safra. On the hardness of approximating minimum vertex cover. *Annals of Mathematics*, 162:2005, 2004.

[4] Richard Karp. Reducibility among combinatorial problems. 40:85–103, 01 1972.

[5] Michal Parnas and Dana Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theoretical Computer Science*, 381(1):183 – 196, 2007.

[6] Wayne Pullan. Optimisation of unweighted/weighted maximum independent sets and minimum vertex covers. *Discrete Optimization*, 6(2):214 – 219, 2009.

[7] D. Tomar. An improved greedy heuristic for unweighted minimum vertex cover. In *2014 International Conference on Computational Intelligence and Communication Networks*, pages 618–622, Nov 2014.

[8] O. Ugurlu. New heuristic algorithm for unweighted minimum vertex cover. In *2012 IV International Conference "Problems of Cybernetics and Informatics" (PCI)*, pages 1–4, Sept 2012.

Table 1: The results of the stochastic local search algorithm compared with the deterministic algorithm to the minimum of the graphs in the left column. The stochastic local search algorithm has an iterations column which denotes the number of iterations it took to find the vertex cover. Time per iteration column and Time denote the time taken for each algorithm to be run once. The algorithms were tested on the complement DIMACS benchmark graphs.

Graph	Minimum	**Stochastic**	Iterations	Time per iteration	**Deterministic**	Time
brock800_1	777	780	354	5ms	782	5ms
brock800_2	776	780	2098	6ms	785	6ms
brock800_3	775	781	2816	6ms	784	6ms
brock800_4	774	781	1431	5ms	785	5ms
c2000.9	1922	1932	4512	897ms	1939	6ms
c4000.5	3982	3986	7692	5068ms	3987	4794ms
MANN_a45	691	705	1	4ms	695	5ms
p_hat15000-1	1488	1489	1819	5ms	1491	5ms