Minimum Vertex Cover

Andre Mao (am5994) Minh Hien Tran (mt3854)
Tony Giannini (aug2102)

Fall 2024

1 Introduction

The *Minimum Vertex Cover* problem is a quintessential NP-Complete problem. A **vertex cover** of a graph is a subset of the graph's vertices such that every edge in the graph is incident to at least one vertex in this subset. The smallest subset of vertices that is a vertex cover is the **minimum** vertex cover.

Most solutions to this problem are brute force solutions where sets of vertices are built and tested. Although this problem can be solved sequentially, runtime should be able to be improved by doing set testing and construction in parallel.

2 Objective

We plan to build a solver for the minimum vertex cover that works with undirected graphs with any amount of nodes and any degree of density. Our first implementation will be purely sequential. Then, we will see what type of performance improvement we can achieve through parallelization.

3 Algorithm

3.1 Overview

To solve the minimum vertex cover, there are a few algorithms that we can test both in sequential and parallel manners. We aim to deliver both sequential and parallel implementations of the Brute Force Algorithm for this class. While the 2-Approximation Algorithm and Greedy Heuristic Search are described below, we plan to implement them in a later iteration of this project.

We will generate graphs of various node counts and densities with a standalone haskell program. This program will output graphs as edge lists to separate files. These files will be read by the solver. The solver will construct adjacency lists from the input and then produce solutions.

3.2 Brute Force Algorithm

We plan to test the Brute Force solution to the Minimum Vertex Cover problem. Given a graph of vertices and edges, we can construct a set of subsets of size 1 to size $|V|$ and test whether or not the each subset creates a valid vertex cover as described in this reference project [1]. Since the subsets are in non-decreasing order the first subset that creates a valid vertex cover, gives us the/a minimum vertex cover.

There are two parts of this algorithm that lend itself to parallelization. The first part is the subset construction. Since we are constructing $2^{|V|}$ subsets, it would make sense to construct these in parallel. We can start by having a spark created with *par* for each recursive call. Then we can 'go parallel only to a certain depth' to further improve performance by avoiding over-saturating the system with sparks.

The second part of this algorithm that can be parallelized is the verification. For each subset, we can run a parallel task see if every edge is attached. Once all processes are done, the smallest subset that satisfies the property is our answer. The first stab at parellizing this component can be creating a spark for each subset with *par*. Next, we can try to divide the set of subsets into roughly equal chunks and create a spark for each chunk. A final optimization might be to terminate work being done on subsets of larger sizes if one with a smaller size is found to satisfy the property.

3.3 2-Approximation Algorithm

If we hope to further optimize the minimum vertex cover algorithm, we can utilize the 2-Approximation Algorithm as described in Chapter 35 of [2]. This algorithm arbitrarily chooses an edge (u, v). Once the edge is chosen, all vertices connected to u and v are removed from the graph until all vertices are reached. By implementing this algorithm, it guarantees a minimum vertex cover that is at most twice the size of the optimal minimum vertex cover in a polynomial-time algorithm.

The part of this algorithm that lends itself to parallelization is the selection and subsequent traversal of edges. We can parallelize the selection of the first edge and the subsequent graph, along with the next selection, and so on and so forth.

3.4 Greedy Heuristic Search

One last algorithm that could be explored is a greedy heuristic search for the minimum vertex cover. As described in [1], we can greedily select vertices that cover the most vertices, as it's likely that by removing as many vertices as possible with each selection, we are able to reach the minimum vertex cover in a more efficient manner.

However, the sequential nature of a greedy search does not lend itself well to parallelization. Since each choice is affected by the previous choice, it seems

that this approach would not be great to explore in a project like this, although feedback on this would be much appreciated.

4 References

[1] https://github.com/sedgwickc/VertexCoverSearch
[2] Introduction to Algorithms, 3rd Edition. Cormen, Leiserson, Rivest, Stein.