Introduction

The Minimum Vertex Cover (MVC) problem seeks to find a subset of vertices for a given graph G such that all the said vertices touch all edges of the graph. This is a well known NP Complete problem that has many real world applications including finding phylogenetic trees based on protein domain information [1] or finding where to place bunkers to cover all cities. For this project, our group was tasked with solving the MVC problem using three different approaches: branch-and-bound, local search, and approximation.

The approximation approach is one that sacrifices accuracy of the result in favor of speed. Generally, approximation algorithms have a provable guarantee that they will be no worse than $p(n) * OPT$, where p(n) is some value greater than or equal to 1 that may depend on the size of the problem and OPT is the optimal solution. For the MVC problem, the approximation approach involves arbitrarily picking an edge in the given graph, then removing is and all neighboring edges from the graph. Then a new edge is arbitrarily picked from the remaining ones and the process is repeated until all the edges have been removed. The vertex cover will then be the nodes corresponding to all the arbitrarily picked edges.

Local Search I is a generic version of hill climbing. Instead of choosing neighbours based on conditions like next vertex has an edge to the last added vertex, the neighbourhood choice is made from a random uncovered edge.In addition to the traditional hill climbing approach, the local search I algorithm has options such as forward stopping when the optimal vertex count is reached, remembering the best solution obtaining so far, growing and shrinking (either direction traversal). The local search approach begins with an initial solution and then tries to obtain the global minima by choosing neighbours and moving to the neighbour with the best eval function. The eval function for the local search approach is the number of edges incident on the vertex that is to be added to the current set of vertices in our current search iteration.

Problem

Consider a graph G = (V,E). Formally, a vertex cover V' is a subset of V such that for each $(u, v) \in E$ it follows that either $u \in V'$ or $v \in V'$. This set V' is said to *cover* the edges of G. For the project at hand, this problem is further complicated by searching for the smallest possible V' that satisfies the above definition. The MVC problem is known to be NP Complete, and as such it requires specialized algorithms such as local search and approximation in order to be solved.

Related Work

Algorithms

Approximation Algorithm:

As mentioned above, the approximation algorithm works by arbitrarily picking an edge in the graph, adding its nodes to the solution, then removing the edge and all of its neighboring edges from the graph, and repeating this process until all edges have been removed. In pseudocode, this looks like the following:

```
MVC_approx(G):
C = { }
E' = E
while E' is not empty:
        remove e in E' and add its nodes to C
        remove all edges neighboring e from E'
return C
```

This algorithm is inspired by that found in CLRS Chapter 35. The algorithm works by choosing an edge arbitrarily, then adding its nodes to the solution and removing all edges neighboring said nodes. This is done because the nodes added to the solution now cover all the edges, so they do not need to be considered. By repeating this process until every edge has been removed from the graph, this guarantees that the solution will cover all edges in the graph by design. This algorithm is provably a 2-approximation algorithm, which means that any solution found will be no worse than twice the size of the optimal solution.

Because of its structuring, it is not particularly complex and thus fast compared to the other algorithm options. But it also does not give exact solutions except for particular cases, which can be disadvantageous depending on the application of it. As seen in the table below, the approximation algorithm runs extremely quickly, but has high relative error.

Future approaches to this algorithm may be to pick the edge and nodes in a smart way so as to remove more edges during each loop.

Local Search Algorithm I:

Initial Solution and approach:

The local search is capable of traversing two ways.

i) When the vertices in the current search iteration form a vertex cover, we try to remove a vertex so as to see if a lesser size vertex cover is possible

ii) When the vertices in current search iteration do not form a vertex cover, we try to add a maximum impact vertex from a random uncovered edge and perform the check for vertex cover once again.

Here the maximum impact vertex is the vertex with most edges incident on it. Random uncovered edge is chosen from the set of edges in the graph that do not have any of its vertices on the current iteration of VC. In order to get a guaranteed VC, we can start with initial solution equal to the total number of vertices and the minimum impact vertices are removed. However, for large graphs this can be quite time consuming - for each VC of specific length almost all other possible permutations are tried. Thus the rate at which the VC size decreases is very low. The optimal strategy is to begin with |V|/2 vertices which have the maximum impact which is done in the method ConstructVC. The current iteration vertices list will grow to add more vertices if a VC is not currently found.

Pseudocode for Local Search I:

```
Algorithm 1: LocalSearch(G,no_of_iterations,k)
Input: graph G = (V, E), no_of_iterations
Output: vertex cover of G if k = no of vertices else optimal vertex coverage obtained so far
Begin
gain(v) := no of edges for each vertex v ∈ G
Circ_deque_len = circular deque of length 100
C := ConstructVC(k); // constructs list of size k with vertices sorted according to no of edges incident on them
While no_of_iterations do
        no_of_iterations <- no_of_iterations-1
        Edge_set = set of edges covered by C
        if Edge_set == total_no_of_edges:
                Circ_deque_len.append(C.length)
                If length of C is the minimum length in Circ_deque_len:
                        Optimal_soln = C
                remove a random Vertex from C;
                Continue
        U,V = vertices of a random uncovered edge
        Append either U/V to C depending on the number of edges incident on U,V
        Return Optimal_soln or C if Optimal soln is empty
```

Local Search Algorithm II:(Simulated Annealing with random restart)

Neighbourhood choice: Choosing a vertex with more number of edges incident to them does not necessarily ensure edge coverage. The second approach allows us to add the vertex with lesser number of edges incident on them with a specific probability. With Simulated Annealing, we can allow exploration at lower temperatures(more chances of adding vertices with lesser vertices incident on them) and at higher temperature, there is more exploitation of the gradient.

Thus the probability of choosing a vertex with lesser number of edges incident on them is: P(C,n) = 1/ (1+ $e\ pow(\triangle E/T)$) where n is the next vertex to be added to current VC vertices. $\triangle E$ is the number of edges additionally covered by adding n. T is temperature.

Pseudocode for Local Search II:

T ← very - high
C ← random_initial_soution
While T do:
    Monotonically decrease T
    n ← random_neigh(C)
    Evaluate $\triangle$E
    Evaluate P(C,n)
    Add n to C with probability P(C,n)

Empirical Evaluation

Approximation

Graph	**OPT**	**Approx.**	**Time (s)**	**Error**	**Graph**	**OPT**	**Approx.**	**Time (s)**	**Error**
Jazz	158	186	0.094	0.18	Star2	4542	6786	508.59	0.49
Karate	14	22	0.00071	0.57	Net Science	899	1224	1.53	0.36
Football	94	110	0.016	0.17	Email	594	834	1.17	0.40
July22	3303	6026	142.50	0.82	Delaunay	703	960	1.03	0.37
Hep-TH	3926	5784	51.21	0.47	Power	2203	3788	16.96	0.72
Star	6909	10536	268.7	0.52	----------	------	----------	----------	---------

Local Search I

File(V,E,Optimal)	**No of iterations, Time for K=|V|/2**	**For K= |V|/2 VC obtained in less than 1000 iterations? T/F, VC length**	**For K= |V|/2 Quality if VC not obtained**	**No of iterations Time for K=|V|**	**For K = |V| VC length**
karate 34,78,14	937 0.361649990	True 14	N/A	983 0.376604080	14
football 115,613,94	952 3.040755033	True 99	N/A	1000 3.118292808 2000 6.935075998 3000 10.36444902	101 99 97
jazz 198,2742,158	973 10.78175282	True 164	N/A	1000 11.67363405 2000 24.90769696 3000 37.15304613 4000 51.93141293 5000 65.98587298	182 175 175 170 169
delaunay n10 1024,3056,703	1000 125.3190948	False 720	2996/3056	1000 100.3215329	972

	2000 244.2097299 3000 374.2991330 4000 511.8474569 5000 620.9045400 5533 678.4559719	False 774 False 783 False 787 False 785 True 784	3046/3056 3053/3056 3055/3056 3055/3056 N/A	2000 250.9891130 3000 394.7654559 4000 523.7575101 5000 633.2892811 6000 797.7224891	963 946 934 929 920

We can see from the above table that VC can be obtained by increasing the number of iterations for larger graphs if k= |V|/2.
This speed of reaching the optimal vertex cover can be drastically improved by adding or removing multiple vertices to the current iteration of vertex cover instead of one vertex at a time. The result for adding/removing multiple vertices will be added in the final report.

Discussion

Conclusion

Bibliography

1. 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.
2. Shaowei Cai, Balance between Complexity and Quality: Local Search for Minimum Vertex Cover in Massive Graphs, Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015).

*This format is solely for the project checkpoint. Once all writing has been completed, the format will be adjusted to ACM.