Algorithm Implementations for Solving Minimum Vertex Cover

Sheng-Yi Cheng
Department of Computational Science and Engineering
Georgia Institute of Technology
Atlanta GA, US
scheng98@gatech.edu

Ting Liao
Department of Computational Science and Engineering
Georgia Institute of Technology
Atlanta GA, US
tliao32@gatech.edu

Chin Wang
Department of Civil and Environmental Engineering
Georgia Institute of Technology
Atlanta GA, US
cwang732@gatech.edu

ABSTRACT

The Minimum vertex cover is a NP-Complete problem in the field of computational study. This paper focuses on the performance of four distinct algorithms which is the Branch and Bound algorithm, two Local Search algorithms (Simulated Annealing with Random Restart and Hill Climbing) and an Approximate algorithm in solving the Minimum Vertex Cover problem. The evaluation is based on the quality of the solution and the runtime of solving the input data graph. The Local Search algorithm provided promising solutions for all the datasets, while the Branch and Bound algorithm was less accurate because of the short cutoff time, 600 seconds. The approximate algorithm showed the highest relative error but it could find a solution in a very quick, as expected from its theoretical understanding.

KEYWORDS

Minimum Vertex Cover, Branch and Bound, Approximation Algorithm, Local Search, Hill Climbing, Simulated Annealing, Random Restart, Qualified Runtime Distributions, Solution Quality Distributions

1 Introduction

The minimum vertex cover (MVC) problem is a NP-complete problem that finding a set of vertices which covers every edge and the number of vertices in the vertex cover is minimized. Since the problem is NP-complete, there is no efficient algorithm for solving the MVC problem. However, there are several strategies which can help us solve the problem in exponential runtime or find a relative optimal solution in a shorter runtime. In this paper, we design different algorithms in order to find the best balance between the runtime and the quality of the solution.

In this paper, we implemented four different algorithms to solve the MVC problem. First, the Branch and Bound algorithm was used for deriving the exact, optimal solution. By using this algorithm, we presented the lower bound which could help us prune the searches. Although computing the exact solution is really expensive on the time consuming, it could eventually return the optimal solution. Second, we implemented two different Local Search algorithms which could return increasingly better feasible solutions quickly. However, they couldn't promise the quality of the return solution. Finally, the Approximation algorithm was implemented to generate the solution in polynomial running time for the worst case.

In the following sections, we will provide detail explanation and pseudo code for each algorithm. Then, after executing all datasets, we will give some conclusions and future works.

2 Problem Definition

An undirected graph G (V, E) consists of a set of vertices V and a set of edges E. Each edge in E consists of a pair of vertices (u, v) $\in$ V. Given an undirected graph G (V, E), a Vertex Cover C is the subset of V such that (u, v) $\in$ E => (u $\in$ C)$\vee$(v $\in$ C), which means that each edge has at least one vertex in C. The MVC is the set C such that the number of vertices in C is minimized.

3 Related Work

Some works in DNA sequencing and computer network security implement minimum vertex cover algorithm to help solve the problems efficiently.

In computer network security, virus attacks are primary issues and computer scientists develop some strategies to protect computer network from attacks. One of the strategy is to simulate servers as nodes in graph and implement minimum vertex cover algorithm. According to experiment results (Filiol et al. 2007), the number of servers which are minimum vertex cover have more significant effects in defending virus attacks and prevent worm propagations efficiently.

In DNA sequencing, Single Nucleotide Polymorphism (SNP) assembly is a primary issue to solve. The researchers simulate sequences of a sample as a node and conflicts between sequences as an edge. The minimum vertex cover problem is used to find minimum possible sequences and the corresponding set will be excluded to reduce the conflicts. (Lancia et al. 2001)

Theoretically, the minimum vertex cover algorithm might not lead to optimal solutions in complicated applications. However, some practical cases mentioned above prove that minimum vertex cover method will efficiently tackle with the problems and improve performance based on experiments results.

In addition, flow shop scheduling problems, are a class of scheduling problems with a workshop in which the flow control shall enable an appropriate sequencing for each job and for processing on a set of machines or with other resources 1,2,...,m in compliance with given processing orders. The proposed methods to solve flow shop scheduling problems can be classified as exact algorithm such as Branch and Bound. The branch-and-bound (B&B) algorithmic framework has been used successfully to find exact solutions for a wide array of optimization problems. B&B uses a tree search strategy to implicitly enumerate all possible solutions to a given problem, applying pruning rules to eliminate regions of the search space that cannot lead to a better solution. (Morrison et al. 2016)

What's more, Hill Climbing is a heuristic search used for mathematical optimization problems in the field of Local search algorithm. Given a large set of inputs and a good heuristic function, it tries to find a sufficiently good solution to the problem. This solution may not be the global optimal maximum. In the above definition, mathematical optimization problems imply that hill climbing solves the problems where we need to maximize or minimize a given real function by choosing values from the given inputs. Travelling salesman problem is a classical local search problem where we need to minimize the distance traveled by the salesman. Heuristic search means that this search algorithm may not find the optimal solution to the problem. However, it will give a good solution in reasonable time. A heuristic function is a function that will rank all the possible alternatives at any branching step in search algorithm based on the available information. It helps the algorithm to select the best route out of possible routes.

4 Algorithms

4.1 Branch and Bound

The idea of the BnB algorithm is to find smaller subsets of the parent graph, G, for partial solutions (vertex covers), while pruning branches that do not satisfy the bounds to the optimal solution. The bounds to the partial optimal solution can be defined in multiple ways, but must be provable to encompass the optimal solution, and are continually updated to lower bound of possible solutions. In the implementation of the BnB algorithm we did, the upper bound solution is the best solution (vertex cover of least size) encountered at any given stage during exploration, while the lower bound solution for the subset graph. Given a parent node graph G = (V, E) and a partial vertex cover C' to the explored section of G, lower Bound(subproblem) = C' + lower Bound(G'). G' is the unexplored graph. The BnB algorithm considers vertices in decreasing order of vertex degree as the most potential candidate solution.

4.1.1 Time and Space Complexity

According to the method we use, it needs exponential time, O(2^{|V|}). Also, the space complexity is O(2V)

4.1.2 Strength and Weakness

Branch and bound can provide the exact solution compared to the other two algorithms, Local Search and Approximation. However, it has the weakness of sacrificing exponential computing time.

4.1.3 Pseudo-code for Branch and Bound

Algorithm 1 Branch and Bound

```
function BnB(G, cutoff, seed)
    VC = $\emptyset$
    v = 0                                                        ▷ Hightest degree
    Frontier = [ (v, 0, (-1, -1)), (v, 1, (-1, -1)) ]  ▷ subproblems(vertex, tuple, (parent tuple, parent state))
    UpperBound = $|V|$                         ▷ initial upper bound = number of nodes in graph
    while Frontier $\neq \emptyset$ do
        $v_i, state, parent$ = Frontier.pop()
        VC $\leftarrow$ $(v_i, state)$
        backtrack = False
        if state == 0 then
            set state = 1 for all neighbors of $v_i$
            VC $\leftarrow$ All neighbors of $v_i$
            $G \leftarrow G$
        else if state == 1 then
            $G \leftarrow G$
        end if
        if G.edges == $\emptyset$ then                              ▷ End of exploring
            if $|VC| < UpperBound$ then
                opt $\leftarrow$ VC
                UpperBound $\leftarrow |VC|$
                BackTrack = True
            end if
        else
            if LowerBound(G) + $|VC|$ <UpperBound then                ▷ Keep exploring
                $v_j$ $\leftarrow$ vertex with highest degree among unexplored nodes
                Frontier $\leftarrow$ Frontier $\cup$ $[(v_j, 0, (v_i, state)), (v_j, 1, (v_i, state))]$
            else
                BackTrack = True
            end if
        end if
        if BackTrack == True *and* Frontier $\neq \emptyset$ then
            NextParent $\leftarrow$ Last Element in Frontier
            if NextParent $\in$ VC then
                index $\leftarrow$ index of NextParent in VC
                for i = index to $|VC|$ do
                    remove VC(i) from VC
                    add VC(i) and its edge back to G
                end for
            else
                reset VC $\leftarrow$ empty
                reset G $\leftarrow$ Original G
            end if
        end if
    end while
end function
```

Figure 1: Pseudo-code for Branch and Bound

4.2 Local Search

The Branch and Bound algorithm can provide the optimal solution, but it needs exponential time. Using the Local Search Algorithm can provide increasingly feasible solutions quickly. The Local Search method can quickly derive an initial solution and move to a neighboring solution with a better return value of the evaluation function.

We implemented two Local Search algorithms to solve the MVC problem. From their output, we would see the gradually improvement of the feasible solutions.

4.2.1 Local Search 1 – SA with Random Restart

This method starts from randomizing the order of the edges to perform a randomly initialized feasible solution. From the initial state, it can move to a better neighbor solution by iteratively removing one of the vertex in a selected edge which is already in the vertex cover. By doing so, the vertex cover can remain valid because we only remove the vertex whose adjacent vertex is in the vertex cover. Also, we decide to remove the vertex with lower degree because lower degree means that the vertex covers much fewer edges than its adjacent vertex. In order to prevent from local optimal solution, we used the concept of simulated annealing (SA) with random restart. To be more specific, we randomly assigned the deleted vertices back to the vertex cover with a gradually decreased probability. Therefore, it increased the search space to find the optimal solution.

4.2.*1.1 Neighborhood Relation*
In this algorithm, we defined our neighborhood relation that after looping through all edges and removing the existing adjacent vertices, the neighbor solution was found. That is, the neighbor solution has the configuration that some of vertices were removed from the current vertex cover.

4.2.*1.2 Evaluation Function*
In the algorithm, we used three methods to ensure the quality and validity. First, removing any vertex from the vertex cover might cause the vertex cover to be invalid. However, we only removed the vertex whose adjacent vertex is in the vertex cover. Thus, our algorithm promised to move from one valid solution to another valid solution. Second, while removing the vertex, we first removed the vertex with a lower degree that it covers much fewer edges than its adjacent vertex. Using this method, it could possibly provide a better solution after removing any vertex. Third, in the end of each iteration, the algorithm would compare the current vertex cover with the previous, best vertex cover to find out the better solution. For example, if the length of the current solution is 20 and the previous, best solution is 25, the current solution would become the best solution and be recorded. If the current solution is worse than the best solution, the algorithm would start a new iteration. From the above methods, our algorithm can ensure the quality and validity of the solution.

4.2.*1.3 SA with Random Restart*
In this algorithm, we employed random restarts in order to prevent the local optimal solution. Each random restart consists of the probability to execute, randomized initial vertex cover and the randomized order of edges. Here we implemented the concept similar to simulated annealing method whose probability decreased by the time. For suitably chosen probability settings (which vary between problem instances, but in many cases can be as high as 0.5) (Hoos et al, 2006), we defined our probability from 25% to 50%. The number of randomly selected vertices which will be added to the vertex cover is defined as 25% (Yang, 2014). By using the probability, we could allow few worsen steps in order to increase the search space. Besides, only randomly taking a part of removed vertices to place back to the vertex cover can make the search space become more diversified. Therefore, using the SA method combining with random restart, we could explore as much neighbor solutions as we can to find the optimal solution.

4.2.*1.4 Time and Space complexity*
In the Local Search 1 method, we implemented a while loop with a for-loop inside it. Generally, the worst case could be O(N^2). However, we would set the cutoff time to 600 seconds. Therefore, it is possible that the runtime of the algorithm become polynomial time complexity which satisfy our anticipation to the local search method. The space complexity would be based on the edges, so it should be O(E) = O(N).

4.2.*1.5 Pseudo-code for Local Search 1*
The following is the pseudo-code for the Local Search algorithm using the simulated annealing with random restart. After we input the data graph, we will get the optimal solution of the minimum vertex cover.

Pseudo-code :

```
// Input Graph G(V, E), cutoff
// Initial setting
currMVC ← all V
bestMVC ← all V
iteration = 0
time = 0
while time < cutoff do
    if iteration > 0 then
        if meet the 0.25 probability then
            pick random number of removed vertices
            and place back to currMVC
        end
    end
    Edges ← randomize the order of the edges
    for edge(u,v) in Edges do
        if u, v both belongs to currMVC then
            // Compare the degree of u, v
            if degree(u) < degree(v) then
                currMVC remove u
            end
            else
                currMVC remove v
            end
        end
    end
    if currMVC < bestMVC then
        // update bestMVC
        bestMVC ← currMVC
    end
    iteration += 1
end
// Output MVC
return bestMVC
```

Figure 2: Pseudo-code for Local Search algorithm using the simulated annealing with random restart

4.2.2 Local Search 2 – Hill Climbing

The idea of the hill climbing is implemented as to take in the heuristic greedy independent cover algorithm as an initial solution, and then keeps swapping points to change the solution state to explore the neighboring search space until the solution is a vertex cover. The neighbor solution that minimizes the cost function, in this case the number of uncovered edges, is iteratively moved to. However, due to the graph is unweighted in the instances. So we do not need to calculate the cost. Moreover, the implementation of the taboo states help reduce the frequency of this algorithm from getting stuck at local maxima. Whereas the taboo state implementation prevents re-searching the same search space. But since this algorithm is still a local search algorithm, there is no guarantee that this algorithm will find an optimal solution and might only find a locally optimal solution. In contrast Branch and Bound gives the guarantee of finding a globally optimal solution given enough time.

4.2.*2.1 Time and Space Complexity*

In this algorithm, we defined our neighborhood relation that we climb to a neighbor solution each iteration. So, our time complexity is O(Edge*Iteration) and space complexity O(|V|+|E|).

4.2.*2.2 Strength and Weakness*

Local Search algorithm using the Hill Climbing concept can quickly find a feasible solution. However, the quality of the solution cannot be guaranteed.

4.2.*2.3 Pseudo-code for Local Search 2*

The following is the pseudo-code for the Local Search algorithm using the simulated annealing with random restart. After we input the data graph, we will get the optimal solution.

Algorithm 1 Local Search: Hill Climbing

```
function LS2(G, cutoff, seed)
    C ← Initial Vertex Cover
    while CurrentTime - StartTime < Cutoff do
        while C is VertexCover of G do
            Store Solution: C* ← C
            Remove vertex from C leading to highest cost loss
            Where Cost(G, C) = $\sum$ w(e) for each uncovered $e \in E$
        end while
        u = vertex from C leading to highest cost loss
        C ← C / u
        Taboo(u) = 0
        Taboo(x) = 1 for x in Neighbors(u)
        v ← endpoint of random uncovered edges: $s.t$ C / v and Taboo(v) == 1
        C ← C ∪ v
        w(e) += 1 for each uncovered $e \in E$
    end while
end function
```

Figure 3: Pseudo-code for Local Search using Hill Climbing

4.3 Approximation

In approximation algorithm, we use maximum degree greedy algorithm mentioned in Reference 1 to implement. The algorithm will choose a node which has maximum degree in the graph and append it in vertex cover list. Then the node will be removed from the graph and the algorithm will continue to find other node with maximum degree in remaining graph until all nodes have 0 degree. In addition to refer original concept, we try to remove the edges which are adjacent with the removed node. According to results, we find that whether remove the edges or not will not result in conspicuous effects.

4.3.1 Pseudo-code for Approximation Algorithm

Pseudo-code :

```
// Input Given G = (V, E)
// Initial setting
Node ← Find Max Degree(G)
VertexCover ← empty list
while $Degree(Node) > 0$ do
    VertexCover ← add Node into the list
    Remove Node and adjacent edges from G
    Node ← Find Max Degree(G)
end
// Output MVC
return VertexCover
```

Figure 4: Pseudo-code for Approximation Algorithm

4.3.2 Approximation guarantee

In research, the approximation ratio of maximum degree greedy algorithm is lnD+0.57 where D is maximum degree of the graph (Delbot et al. 2010). Therefore, when the graph is large, the approximation bound will be large and the solutions developed from approximation might deviate from optimal solution a lot. In this project, we can observe that the errors in small-scale cases is much less than large-scale cases. In other words, this approximation ratio is quite reasonable.

4.3.3 Time & Space Complexity

The algorithm keeps iterating when a node has at least one degree. That is, at least one node has one neighbor node. Therefore, at most O(V) nodes will be checked and it takes O(V). In the while loop, when a greedy node is appended in a vertex cover list, the node and its neighbors will be deleted. In other words, the algorithm tries to find its possible neighbors and the amount of neighbors is at most O(V). Consequently, the time complexity is O(V*V) = O(V^2). Vertex cover list contains the nodes which can cover all nodes by adjacent edge. Therefore, the list at most includes O(V) nodes, so the space complexity is O(V).

4.3.4 Strength & Weakness

The running time in worst case is polynomial. The quality of solutions can be guaranteed in a specific range. The solutions might not be optimal solutions.

5 Empirical Evaluation

The algorithms were implemented in the background of Python 3.7. To ensure the accuracy of the results, all programs were tested on the MacOS system with the version Catalina 10.15.7, which has a 2.7 Ghz Dual –Core Intel i5 processor, 8 GB RAM.

The final results were generated from all 11 datasets (except two dummy datasets). For the two local search algorithms, each of them was executed by using 10 different random seeds and the performance was averaged from the 10 results. The performance was evaluated based on the runtime of finding the optimal solution within a given cutoff time.

5.1 Branch and Bound Results

Branch and Bound has the most optimal among all the result in our work. However, it is time-consuming and do not generate many results in the cutoff time due to the exponential time complexity. It is a fair trade-off. In our opinion, if we raise the cutoff time. We could get the promising result. Moreover, the lower bound we got is closer to the true optimal solution compare to other algorithms. Interestingly, the relative error does not have the positive correlation to the graph size. The worst relative error is in star.graph which is 6.7%. The best relative error is 0 which means the result match the optimal solution provided in the project description. It is quite surprised to our expectation. The relative error from other instance graph vary from 0.2% to 6%. We evaluate we have qualified performance in branch and bound.

Nevertheless, there are still some improvement we can do to modified in the future. We want to shorten the cutoff time as well as to keep the good performance. The following is the lower and upper bound:

Delaunay: Lower bound: 740/Upper bound: 739, Football: Lower bound: 96/Upper bound: 95, Jazz: Lower bound: 160/Upper bound: 158. Other graph only has one candidate solution.

Branch and Bound				
Dataset	Opt	VC value	RelErr	Time
email	594	605	0.0185	0.74
jazz	158	158	0.0000	274.85
karate	14	14	0.0000	< 0.01
football	94	95	0.0106	0.40
as-22july06	3303	3312	0.0027	126.91
hep-th	3926	3947	0.0053	48.91
star	6802	7366	0.0829	112.30
star2	4542	4677	0.0297	128.23
netscience	899	899	0.0000	1.56
delaunay_n10	703	739	0.0512	4.18
power	2203	2272	0.0313	12.62

Table 1: Branch and Bound Results

5.2 Local Search 1 Results

The graphs of Qualified Runtime Distributions (QRTDs) and Solution Quality Distributions (SQDs) are shown as Figure 7 and Figure 8. Also, Table 2 represent the average value of the VC value and the Relevant Error.

For QRTDs, the x-axis represents the runtime with the unit in seconds and the y-axis is the rate of the successful algorithm execution which solves the problem. We set the relative solution quality as 3.0%, 3.1%, 3.5% and 3.6% to plot the power.graph and 9%, 10%, 11% and 12% to plot the star2.graph. We can see the plot clearly shows that with lower relative quality, it needs longer runtime to find the successful solution. Besides, SQDs are based on several fixed runtime and the changing relative quality value. To elaborate, we can see from the graphs that with longer runtime, it can allow larger relative solution quality.

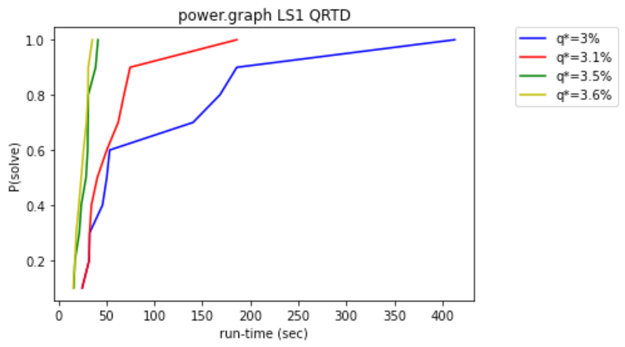


Figure 5: QRTDs plot of power.graph using LS1

Local Search 1				
Dataset	Opt	VC value	RelErr	Time
email	594	605	0.0185	212.73
jazz	158	158	0.0000	10.03
karate	14	14	0.0000	< 0.01
football	94	94	0.0000	0.73
as-22july06	3303	3830	0.1596	564.47
hep-th	3926	3988	0.0158	495.67
star	6802	7474	0.0988	584.12
star2	4542	4837	0.0649	589.22
netscience	899	899	0.0000	102.19
delaunay_n10	703	726	0.0327	122.57
power	2203	2264	0.0277	347.82

Table 2: Local Search 1 Results

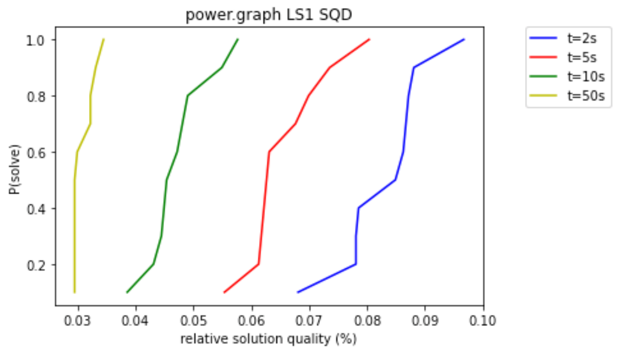


Figure 6: SQD plot of power.graph using LS1

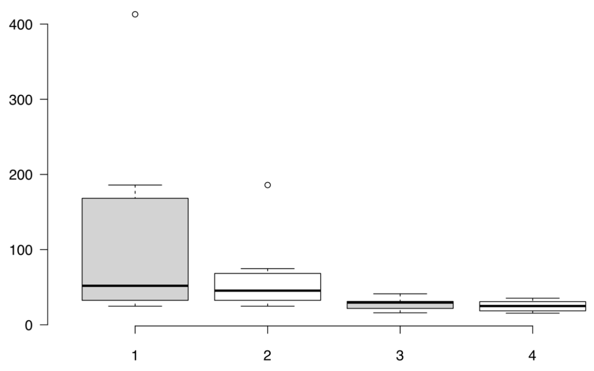
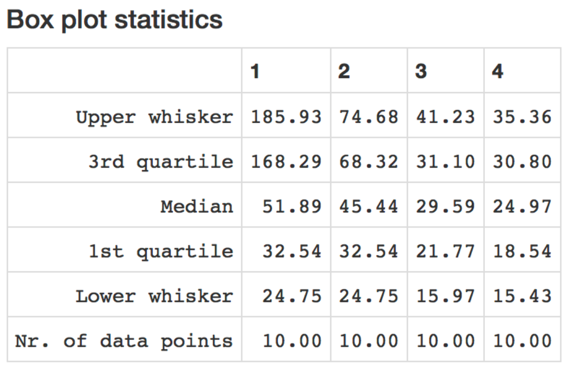


Figure 7: boxplot for power.graph using LS1



Box plot statistics

	1	2	3	4
Upper whisker	185.93	74.68	41.23	35.36
3rd quartile	168.29	68.32	31.10	30.80
Median	51.89	45.44	29.59	24.97
1st quartile	32.54	32.54	21.77	18.54
Lower whisker	24.75	24.75	15.97	15.43
Nr. of data points	10.00	10.00	10.00	10.00

Figure 8: boxplot information for power.graph using LS1

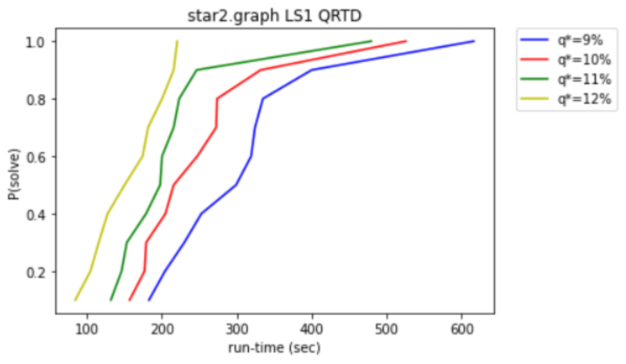


Figure 9: QRTDs plot of star2.graph using LS1

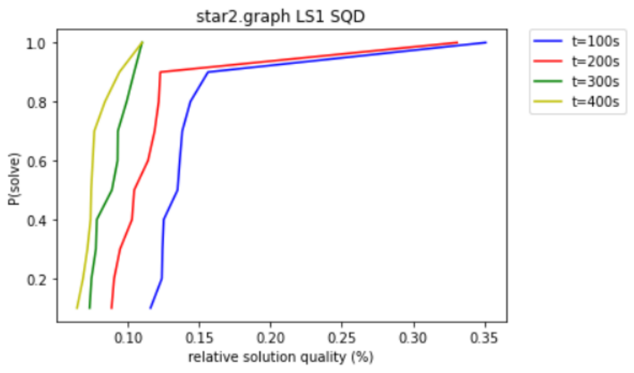


Figure 10: SQD plot of star2.graph using LS1

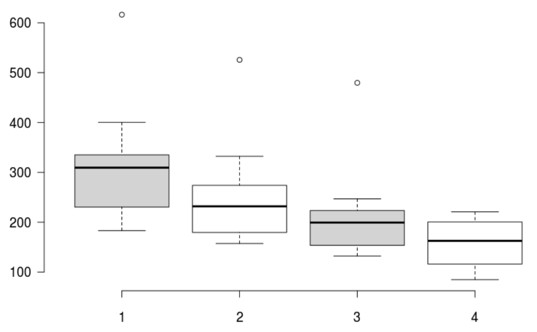
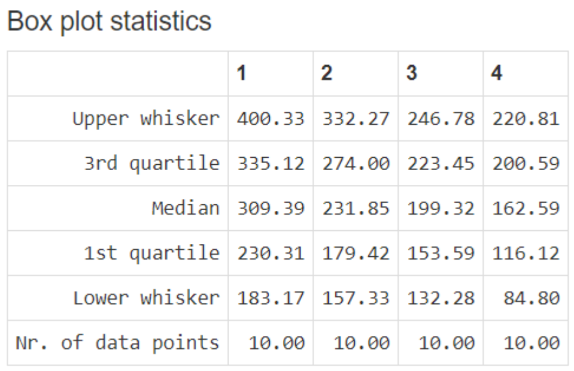


Figure 11: boxplot for star2.graph using LS1



Box plot statistics

	1	2	3	4
Upper whisker	400.33	332.27	246.78	220.81
3rd quartile	335.12	274.00	223.45	200.59
Median	309.39	231.85	199.32	162.59
1st quartile	230.31	179.42	153.59	116.12
Lower whisker	183.17	157.33	132.28	84.80
Nr. of data points	10.00	10.00	10.00	10.00

Figure 12: boxplot information for star2.graph using LS1

5.3 Local Search 2 Results

The result generates from LS2 hill climbing is approaching to the optimal solution. However, the result is not as promising to BnB result. In the result of SQTD line chart from LS2, we can oberve that as the solution quality q*(%) rising, all the lines go up. With the higher solution quality, the lines rise more intensively. However, the q* we choose in these two graph is different (3-6% & 13-16%). We assume the local search results are not really promising as branch and bound. On the other hand, we do save a lot of time during computing. Ideally, the result of SQD should follow the time interval grows up, the solution quality should go down and become a vertical line in the end. In our power.graph result, we do observe this outcome. But the slope of red line (t=20) do not perform well in the chart. A reasonable explanation is that because of random initialization, there is a trace do not generate a normal distribution graph. In that scenario, we do not get an average output form that. On the other hand, we can see that the solution quality sudden raise in the upper part of all the lines in star2.graph. We supposed that there might be a trace to not perform well during random initialization, thus, after sorting the vertex cover of all traces. We got an abnormally higher vertex cover result and cause the performance in our chart. The lower bound and upper bound is presented as follow:

Star2: Lower bound: 5173/Upper bound: 5038, Star: Lower bound: 7581/Upper bound: 7525, Power: Lower bound: 2430/Upper bound: 2268.

In the two box plots from local search 2 hill climbing, we can find that in the power.graph, the left column is extremely different from other three. We consider that the start time window for random initialization, it is reasonable to have a long-range computing time to optimize the vertex cover. We can see that the upper and lower whiskers shrink in the next three column in the box plot of power.graph. On the other hand, the box plot of star2.graph also has the same trend as we mentioned in the power.graph. The box and whiskers all shrink followed by the advance of the time window.

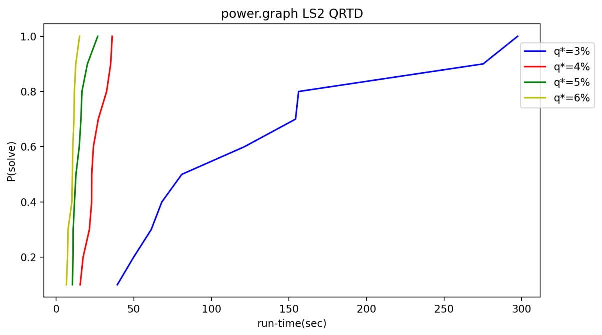


Figure 13: QRTDs plot of power.graph using LS2

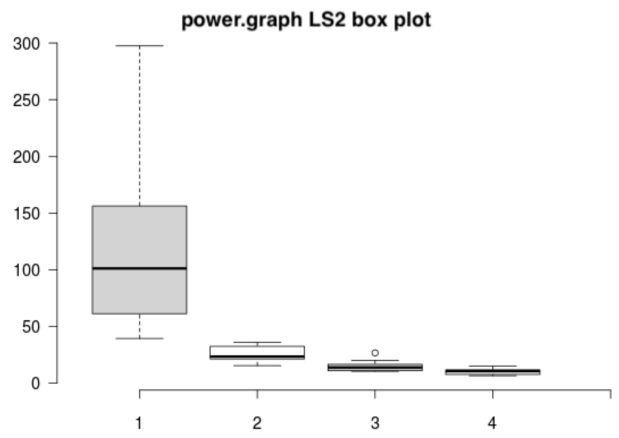


Figure 15: boxplot for power.graph using LS2

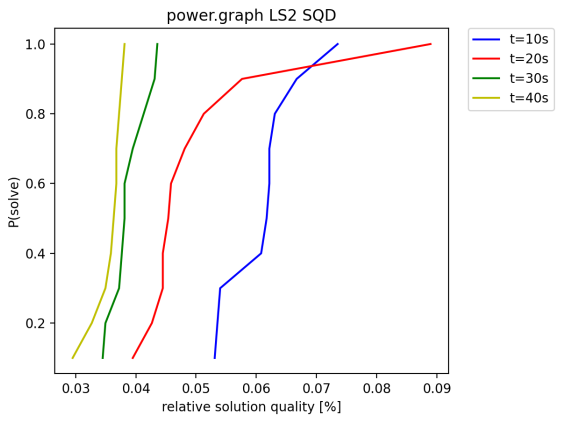
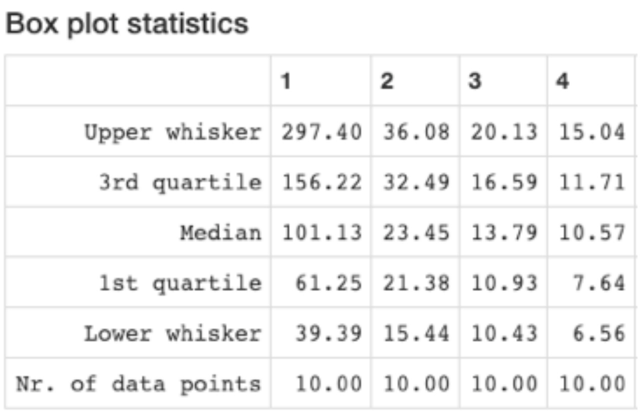


Figure 14: SQD plot of power.graph using LS2



Box plot statistics

	1	2	3	4
Upper whisker	297.40	36.08	20.13	15.04
3rd quartile	156.22	32.49	16.59	11.71
Median	101.13	23.45	13.79	10.57
1st quartile	61.25	21.38	10.93	7.64
Lower whisker	39.39	15.44	10.43	6.56
Nr. of data points	10.00	10.00	10.00	10.00

Figure 16: boxplot information for power.graph using LS2

Local Search 2				
Dataset	Opt	VC value	RelErr	Time
email	594	604	0.0168	288.23
jazz	158	158	0.0000	8.79
karate	14	14	0.0000	< 0.01
football	94	94	0.0000	0.61
as-22july06	3303	3758	0.1378	566.55
hep-th	3926	3997	0.0181	445.47
star	6802	7527	0.1066	488.89
star2	4542	5045	0.1107	295.38
netscience	899	899	0.0000	61.25
delaunay_n10	703	726	0.0327	305.97
power	2203	2266	0.0286	283.22

Table 3: Local Search 2 Results

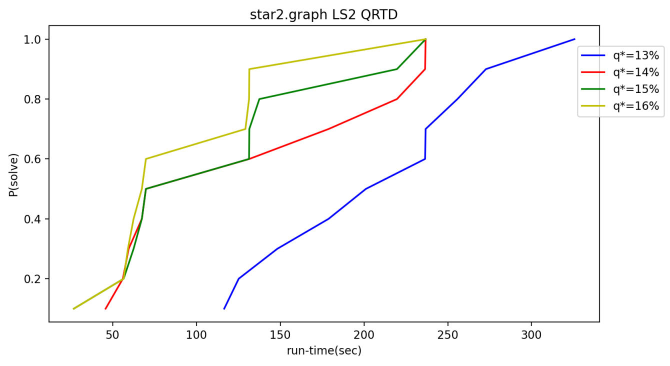


Figure 17: QRTDs plot of star2.graph using LS2

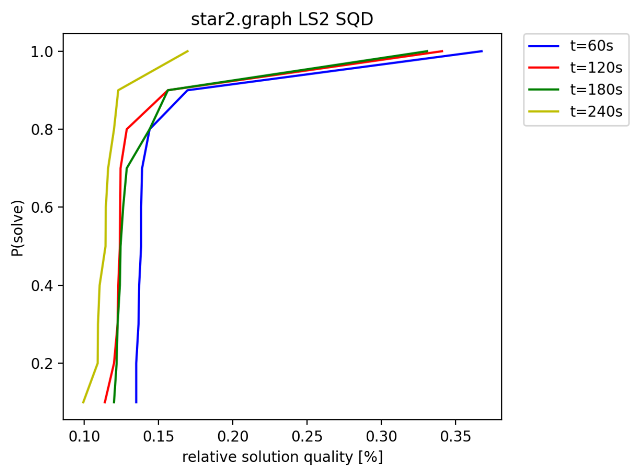


Figure 18: SQD plot of star2.graph using LS2

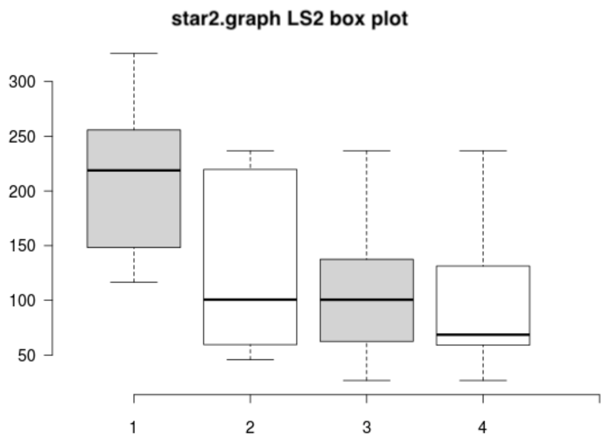
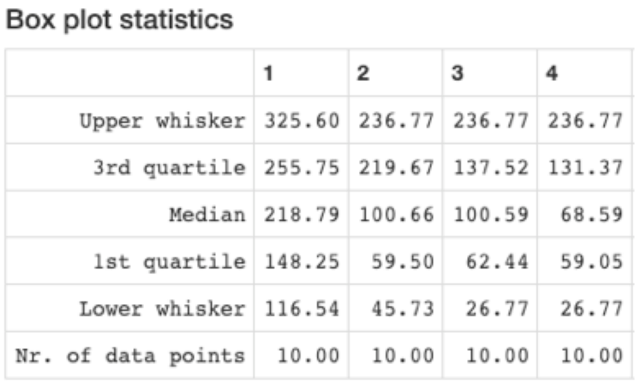


Figure 19: boxplot for star2.graph using LS2



Box plot statistics

	1	2	3	4
Upper whisker	325.60	236.77	236.77	236.77
3rd quartile	255.75	219.67	137.52	131.37
Median	218.79	100.66	100.59	68.59
1st quartile	148.25	59.50	62.44	59.05
Lower whisker	116.54	45.73	26.77	26.77
Nr. of data points	10.00	10.00	10.00	10.00

Figure 20: boxplot information for star2.graph using LS2

5.4 Approximation Results

In approximation algorithm, the lower bound that we derive from the results is minimum vertex cover in each trace file. Table 4 shows that best lower bound is far from optimal solution about 0% and the worst lower bound is far from optimal solution about 8.9%. In the worst case, the dataset has the most amount of vertex cover in optimal solution. For example, the optimal solution of star.graph is 6802 and it has 8.9% error. We can conclude that if the dataset in graph is large, the approximation algorithm has more chances to deviate from optimal solution a lot.

Approximation				
Dataset	Opt	VC value	RelErr	Time
email	594	602	0.0134	250.91
jazz	158	158	0	0.08
karate	14	14	0	< 0.01
football	94	94	0	0.11
as-22july06	3303	3310	0.0021	513.97
hep-th	3926	3939	0.0033	278.33
star	6802	7408	0.089	190.27
star2	4542	4670	0.028	377.06
netscience	899	899	0	0.75
delaunay_n10	703	730	0.0384	232.01
power	2203	2265	0.0281	248.73

Table 4: Approximation Results

6 Discussion

We evaluated the performance of our four implemented algorithms by considering their run times and the relevant error. For local search, we further considered the runtime distribution of each algorithm. By doing so, we could clearly see the result from those plots and proved our previous statement of the different characteristic of each algorithm.

Considering the runtime, the Approximation algorithm provided the best performance. The performance of branch and bound algorithm had the best, exact solution but with the worst run times. The Local Search algorithm should produce an optimal solution within a reasonable runtime. However, based on the comparison of those different tables, we could find that the Local Search solutions have the larger relevant error while implementing with larger graph. The reason is that larger graph might induce the search to be stuck in several local optima and thus cause the solution to be worsen. The Approximation algorithm produced results in the fastest runtime.

For future improvement, it is important to find another method or modify our functions to prevent being stuck in the local optima. Probably utilizing the concept of Tabu search to store a temporary memory in order to backtrack to the previous location when the search become worsen. After fixing the existing problem, our four algorithms could provide users to solve the Minimum Vertex Cover problem effectively.

7 Conclusion

In conclusion, different types of algorithms have its own advantages and disadvantages. To make the decision of using which algorithm to solve the MVC problem should be based on two different aspects, the accuracy of the solution or the runtime. Generally, if the target is to achieve the highest accuracy without any concern on the total runtime, the Branch and Bound method would be the optimal choice. However, if the accuracy is not placed in the first role and a short executing time is required, then the Approximation algorithm would be better. Finally, unlike the other two methods which have obvious advantages and disadvantages, the Local Search algorithms provide a balance between accuracy and speed.

From our research, it clearly shows stable evidence with plot graphs and tables for each algorithm. It could benefit users on choosing which algorithm to solve the problem. The future development on this research may implement the rest of local search strategies. There are several useful approaches that this paper didn't put emphasis on such as Stochastic Local Search, Tabu Search and Iterated Local Search. More comparison on different algorithms could make the research of implementing this problem more robust.

REFERENCES

[1] Xiuwei Chang, 2020, Lecture Notes of CSE 6140 Computational Science and Engineering Algorithms

[2] Xin-She Yang, 2014. Nature-Inspired Optimization Algorithms, Chapter 1

[3] Holger H. Hoos, Edward Tsang. 2006. Foundations of Artificial Intelligence

[4] Francois Delbot and Christian Laforest. Analytical and experimental comparison of six algorithms for the vertex cover problem. Journal of Experimental Algorithmics (JEAN), 15:1-4.

[5] Eric Filiol, Edouard Franc, Alessandro Gubbioli, Benoit Moquet and Guillaume Roblot, 2007. Combinatorial Optimisation of Worm Propagation on an Unknown Network, Proc. World Acad. Science, Engineering and Technology, Vol 23

[6] G. Lancia, V. Afna, S. Istrail, L. Lippert, and R. Schwartz, 2001. SNPs Problems, Complexity and Algorithms, ESA 2002, LNCS 2161, pp. 182-193, 2001. Springer-Verlag

[7] D.R. Morrison, S.H. Jacobson, J.J. Sauppe, E.C. Sewell. 2016. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. Discrete Optim., 19, pp. 79-102

[8] Shaowei Cai, Kaile Su, Chuan Luo, and Abdul Saar. 2013. NuMVC: An efficient local search algorithm for minimum vertex cover. Journal of Artificial Intelligence Research 46, 687–716.