Final Project Report
17FALL CSE6140 Final Project Team 3

Zhuonan Li
School of Industrial and Systems Engineering
Georgia Institute of Technology
leon.li@gatech.edu

Shuho Chou
College of Computing
School of Computational Science and Engineering
Georgia Institute of Technology
benchou1919@gatech.edu

Xinyao Qian
College of Computing
School of Computational Science and Engineering
Georgia Institute of Technology
xinyaoqian@gatech.edu

1 INTRODUCTION

The Minimum Vertex Cover(MVC) is an NP-Complete problem. The problem is to find a set with minimum number of vertices that each edge has at least one point in the set, which is defined as a vertex cover. To deal with it, we implement four algorithms. **Branch-and-Bound** searches an accurate minimum vertex cover for the given graph. **Approximation** finds a reasonable answer in given limited time. **Hill Climbing** greedily removes all the vertices it can to reach a local minima, and actually performs good, with the worst result being only 5% error. **Simulated Annealing** will tolerate some some worsening moves in the beginning, and lower the probability to tolerate along time, and finally get the results that only enhance the result. The worst result it gets is 8.78% on star.graph.

2 PROBLEM DEFINITION

Formally we define the problem in mathematicl formation. Given a graph $G = (V, E)$ denoting the set of vertices and edges, a vertex cover is a subset $C \subseteq V$ such that $\forall(u, v) \in E : u \in C \vee v \in C$. And the minimum vertex cover is to find the minimum $|C|$ that satisfy a vertex cover.

3 RELATED WORK

The minimum vertex cover problem is one of fundamental NP-hard problems in the combinatorial optimization[8] . Therefore, in order to solve it with optimality guarantee, one has to use branch-and-bound types of algorithms to enumerate all possible solutions. To find a minimum vertex cover is very difficult, but to find an alternative is easier. Lots of polynomial time algorithms has been proposed: the Maximum Degree Greedy (MDG) algorithm [5]; the Depth First Search (DFS) algorithm [9], the Edge Deletion (ED) algorithm [8], the ListLeft (LL) algorithm [1], the ListRight (LR) algorithm [6], the Iterated Local Search algorithm [10] and etc. These algorithms have approximation ratios lying between 2 and Δ (the maximum degree of the graph) and the approximation ratio of 2 is the best-known worst-case one[7].

Also, there are many people looking into the parameterized vertex cover problem. Given a graph G and k, the parameterized vertex cover problem is to find a vertex cover of G with at most k vertices[4]. This is the decision problem version of the vertex cover problem. By looking into some properties and introducing new techniques, Chen and Kanj obtained an improved algorithm with time complexity $O(1.2852^k + kn)$[3]. Chandran and Grandoni improves the time complexity to $O(1.2745^k k^4 + kn)$[2] and Chen et al. further improve it to $O(1.2738^k + kn)$.[4]

4 ALGORITHM

4.1 Branch and Bound

4.1.1 Description. The branch-and-bound algorithm uses heuristic algorithm to obtain the initial upper bound and 2-approximation algorithm to obtain the lower bound for the subproblems. The algorithm follows a DFS manner when branching. During branching step, we select the vertex with the most uncovered neighbors to be the branching vertex. Then the algorithm creates two children, setting the branching vertex to be either selected or not selected. If a branch have a lower bound that is worse than the current best solution, then it will not get explored further. If a branch results in a better vertex cover, it replaces the current best solution. The algorithm will terminates when no more branches to explore, or the running time exceeds the cutoff time threshold.

4.1.2 Pseudocode. See Algorithm 1

Algorithm 1 Branch and bound

procedure BNB(Graph $G = (V, E)$, cutoff_time)
  if current_time ≥ cutoff_time **then**
    WriteSolution()
    EXIT
  end if
  if infeasible **then**
    return INFEASIBLE
  end if
  if current_used_vertex + LB ≥ current_best **then**
    return PRUNED
  end if
  $v \leftarrow$ HighestUncoveredDegree()
  $v.used \leftarrow true$
  BNB(G)
  $v.used \leftarrow false$
  BNB(G)
end procedure

4.1.3 Time and Space Complexity. Time Complexity is $O(2^k)$, where k is the maximum depth in our branch-and-bound tree. In our case, $k \leq 2 * |E|$ since we use the 2-approximation algorithm to obtain an initial upper bound.

Space Complexity is $O(|V| + |E|)$ because we only need to maintain the current vertice selection and uncovered edges.

4.2 Approximation

4.2.1 Description. In order to solve this MVC problem in a reasonable time with the outcome approximate the actual minimum vertex cover. Some heuristics with approximation guarantee are provided. Here, we choose the method based on depth-first-search method. In this algorithm, a spanning tree is developed given a starting position, all the nodes excluding the leaf nodes of this spanning tree will be taken into the solution of vertex cover. The apprximation ratio of this algorithm is bounded by 2. The depth first search is conducted by starting from node 0, but when a random seed is imported, the starting point is chosen randomly, therefore, we can try to use different random seeds and see their performance as well as pick up the best solution.

4.2.2 Pseudocode. see Algorithm 2

Algorithm 2 Depth First Search Approximation

function DFS(Graph $G = (V, E)$)
  $T \leftarrow$ DFS Spanning Tree of G from a vertex r
  Let $I(T)$ be the set of nonleaves vertices of T
  return $I(T) \bigcup \{r\}$
end function

4.2.3 Time and Space Complexity. The time complexity of this algorithm is similar to a depth-frist-search complexity, which is $O(|V| + |E|)$. And finally, the non-leaf nodes will be counted in the solution set. The space here used is $O(|V|)$, since the space used in the process of developing spanning tree is $O(|V|)$ and the recursice call stack will be $O(|V|)$.

4.3 Local Search - Hill Climbing

4.3.1 Description. The basic idea of hill climbing is that it searches from a start, and iteratively changes a single element in the solution, preserve the new change if it produces a better result, until there is no further changes can be made. In implementation, the algorithm greedily search from the lowest degree vertex, and see if it can be removed from the original vertex cover, and still be a valid vertex cover by seeing if the vertices connected to the vertex are covered by the vertex cover. To get a better result, I initialize the vertex cover by using Edge Deletion as proposed in [7]. For all the edges inside the graph, pick one edge (u, v) and add u, v to the vertex cover, and then remove all the edges connected to u and v. It repeats the above mentioned operations until the graph has no edge at all.

Since this algorithm can run very fast (5 seconds for the biggest graph), we can start with different initialization, and get better results, so this algorithm will be called over and over until reaching cutoff time, or reaching some other thresholds.

4.3.2 Pseudocode. See Algorithm 3.

Algorithm 3 Hill Climbing

function HillClimbing(Graph $G = (V, E)$, $cutoff\_time$)
  $VC \leftarrow$ EdgeDeletion(G=(V, E))
  Sort vertex cover VC by degree
  for each vertex v in VC **do**
    if $elapsed\_time > cutoff\_time$ **then**
      break
    end if
    $neigh\_v \leftarrow$ neighbors of v
    if $neigh\_v$ contained by $VC - \{v\}$ **then**
      $VC \leftarrow VC - \{v\}$
    end if
  end for
end function

function EdgeDeletion(Graph $G = (V, E)$)
  $C \leftarrow \emptyset$
  while $E \neq \emptyset$ **do**
    select (u, v) from E
    $C \leftarrow C \bigcup \{u, v\}$
    $V \leftarrow V - \{u, v\}$
  end while
  return C
end function

4.3.3 Time and Space Complexity. For time complexity, first we discuss about the Edge Deletion algorithm, it takes $O(|E|)$ to iterate through all the edges, and $O(|V|)$ to see all the vertices, and thus has a complexity of $O(|V| + |E|)$. And for the hill climbing part, we search through all the vertices in the calculated by Edge Deletion, and see if the edges connected are covered by the remained set. We know that for each edge, we can use $O(1)$ to search if it is in the cover set if we use *unordered_set* as the data structure. And we know that at most we need to search for $2 \times |E|$ edges (if all the vertices are chosen in vertex cover), so the total time complexity is $O(|E|) \times O(1)$. And of course, before doing hill climbing, we need to sort the vertex cover, which takes $O(V \log V)$, so with all things combined, the total complexity will be $O(|V| \log |V| + |E|)$.

For space complexity, in Edge Deletion, I only need a set of vertices to indicate the new cover set, which is $O(|V|)$. Whereas in the hill climbing part, I need to store all the edges, which leads to $O(|E|)$, so a total space complexity of $O(|V| + |E|)$.

4.4 Local Search - Simulated Annealing

4.4.1 Description. The basic idea of simulated annealing is that we select a neighbor candidate solution at random, and decide to take a move that worsen the result with a probability. And this probability will go down in time, as we want more stable results at the end, and it is similar to the cool down of temperature, and thus is called simulated annealing. This algoritm tends to do random search at first because of the high temperature, and will search for better solution after the temperature is cool down. The algorithm will run until the temperature is below some preset threshold, or running out of time.

4.4.2 Pseudocode. First I initialize the vertex cover by greedy choice, namely, each time I pick a random edge, and then delete all the edges that are connected to either of the two vertices, until there is no edge left. And in the process, I also keep track of *cost*, which is the difference between the degree and the number of vertices connected that are chosen in the vertex cover. This cost means the effort to remove this vertex. So initially, I remove all the vertices in the vertex cover with cost 0.

For each iteration, we remove all the vertices with 0 cost, and randomly pick one node to remove until it is not a complete cover. After that, we remove a vertex with minimum cost, and add a vertex back randomly. We compare the cost of this one and the previous one, if it is smaller, we substitute it, otherwise, we decide to change it with probability given from temperature and the difference of cost.

Algorithm 4 Simulated Annealing

function SimulatedAnnealing($G = (V, E), cutoff$)
    $VC \leftarrow$ VertexCoverInitialize()
    $VC' = VC$
    while $elapsed < cutoff$ and $temp > quit\_temp$ **do**
        while VC' is vertex cover **do**
            random remove
        end while
        $u \leftarrow$ vertex in cover with minimum cost
        $VC' \leftarrow VC' - \{u\}$
        $v \leftarrow$ randomly pick one vertex in cover
        $VC' \leftarrow VC' \bigcup \{v\}$
        if $costVC' < costVC$ **then**
            $VC = VC'$
        else
            $VC = VC'$ with probability $e^{\frac{costVC' - costVC}{temp}}$
        end if
        $temp* = decay$
    end while
end function

4.4.3 Time and Space Complexity. For time complexity, at each iteration, it randomly removes all the edges that are useless, until no nodes can be removed, which takes $O(|V|^2)$ because there are at most $O(|V|)$ nodes that can be removed, and each node takes $O(|V|)$ to be removed. However, in practice the nodes removed each time is $O(1)$ in average, which gives $O(|V|)$ time complexity. And then deleting smallest cost node and adding random node both takes $O(|V|)$. As for the temperature condition, it takes $O(1)$ to compute, and thus the total complexity of each iteration will be $O(|V|)$ in average.

For space complexity, to store the graph we need $O(|V| + |E|)$, and for the solution set and all the other variables, we need at most $O(|V|)$. So the total space complexity will be $O(|V| + |E|)$.

5 EMPIRICAL EVALUATION

5.1 Branch and Bound

Environment

- Processor: 2.3 GHz Intel Core i5
- Memory: 16GB 2133MHz LPDDR3

5.1.1 Comprehensive Table. The comprehensive results are obtained by the branch and bound algorithm without initial heuristic. We set a cutoff time of 10 minutes and it's clear that the branch and bound algorithm cannot handle the four largest graphs. And from our running log, only **karate** case has finished enumerating all possible branches. That is why even for a slightly bigger graph **football**, the solution at the cutoff time is very closed to the optimal and it actually comes very early.

There is another interesting phenomenon that the algorithm reach optimal for a larger graph, **netscience**, than **football** graph. It is because of the branching strategy that we use, which is finding the vertex with maximum uncovered neighbors. Also, by using DFS strategy, we can get a solution from branch and bound for not too large graph. We've tested this while we were developing the algorithm and compare DFS strategy to BFS exploring strategy.

Dataset	Time(s)	MVC	Optimal	RelErr
jazz	0.47	159	158	0.0063
karate	0	14	14	0
football	0.35	95	94	0.0106
as-22july06	-	-	3303	-
hep-th	-	-	3926	-
star	-	-	6902	-
star2	-	-	4542	-
netscience	7.13	899	899	0
email	6.13	605	594	0.0185
delaunay_n10	5.53	737	703	0.0483
power	153.63	2277	2203	0.0336

Table 1: BNB without initial heuristic solution

Then we add a heuristic to be the first initial upper bound. Theoretically, it should help speed up the bounding process. By comparing Table 1 and Table 2, we can see that only the solution time is faster, but the best solution by the time the algorithm gets cutoff is not improving. However, observing that for the four large graph, the heuristic does not provide good approximation to the optimal vertex cover. Therefore it can explain why the result is not improving too much.

Dataset	Time(s)	MVC	Optimal	RelErr
jazz	0.19	159	158	0.0063
karate	0	14	14	0
football	0.04	95	94	0.0106
as-22july06	0.02	6260	3303	0.8952
hep-th	0.01	5768	3926	0.4691
star	0.03	10406	6902	0.5076
star2	0.05	6870	4542	0.5125
netscience	6.99	899	899	0
email	5.99	605	594	0.0185
delaunay_n10	5.33	737	703	0.0483
power	153.46	2277	2203	0.0336

Table 2: BNB with initial heuristic solution

Lastly we replace the heuristic by one iteration of our Hill Climbing algorithm which provides more promising approximation results. We can see that, from Table 3, all the time gets shorter, especially for **netscience** graph, which is a middle-size graph. In conclusion, a better initial upper bound do helps improving the solution or speed up the process.

Dataset	Time(s)	MVC	Optimal	RelErr
jazz	591.42	158	158	0
karate	0	14	14	0
football	0.04	95	94	0.0106
as-22july06	1.87	3312	3303	0.0027
hep-th	2.01	3946	3926	0.0051
star	5.8	7261	6902	0.0520
star2	2.72	4793	4542	0.0112
netscience	0.08	899	899	0
email	5.39	605	594	0.0185
delaunay_n10	4.77	737	703	0.0483
power	135.61	2277	2203	0.0336

Table 3: BNB with 1 iteration of hill climbing

5.2 Approximation

Environment

- Processor: 2.3 GHz Intel Core i5
- Memory: 16GB 2133MHz LPDDR3

5.2.1 Comprehensive Table. The comprehensive results are obtained by running this algorithm multiple times with different random seed. Here, the seed will randomly choose the starting point and reult in different spanning trees. The solution will also vary given different shape of spanning trees, and the table here presents the best solution found uder multiple seed trials.

Dataset	Time(s)	MVC	Optimal	RelErr
jazz	1.848	182	158	0.152
karate	0.0	15	14	0.0071
football	0.32	104	94	0.106
as-22july06	113.226	4422	3303	0.339
hep-th	6.044	5404	3926	0.376
star	132.991	10388	6902	0.505
star2	316.497	5262	4542	0.158
netscience	1.667	1187	899	0.320
email	2.585	713	624	0.143
delaunay_n10	0.947	859	703	0.222
power	5.201	3366	2203	0.528

Table 4: Result of Approximation based on DFS

As we can see from the table above, the results is better than the former results given in the mid check.

5.3 Local Search - Hill Climbing

Environment

- Processor: 2.3 GHz Intel Core i5
- Memory: 16GB 2133MHz LPDDR3

5.3.1 Comprehensive Table. The comprehensive results are obtained by running 10 different seeds for 10 different runs, and the average is taken. I will not run it for complete 10 minutes, rather, I set a threshold on how long the last update is, and how many times have I run since the last update to determine whether to stop.

Dataset	Time(s)	MVC	Optimal	RelErr
jazz	1.04	159	158	0.0044
karate	0	14	14	0
football	0.07	94	94	0
as-22july06	50.90	3306	3303	0.0010
hep-th	69.82	3943	3926	0.0045
star	115.31	7260	6902	0.052
star2	72.8	4761	4542	0.048
netscience	0.40	899	899	0
email	34.31	606	594	0.0210
delaunay_n10	38.83	740	703	0.0525
power	54.2	2261	2203	0.0265

5.3.2 Quality Run Time Distribution. For the figures below, in **power.graph**, we can see that it reaches a cover within 3.4% difference in only about 5 seconds, and 90% of the running can reach 3% within 30 seconds. As for **star.graph** it takes 5 seconds to get to a vertex cover within 7% error. This result is quite good compare to the others, and we will discuss the comparison between two local searches in discussion.

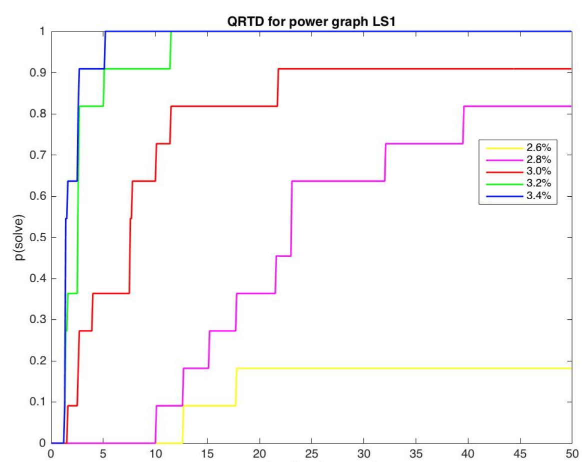

Figure 1: QRTD power LS1

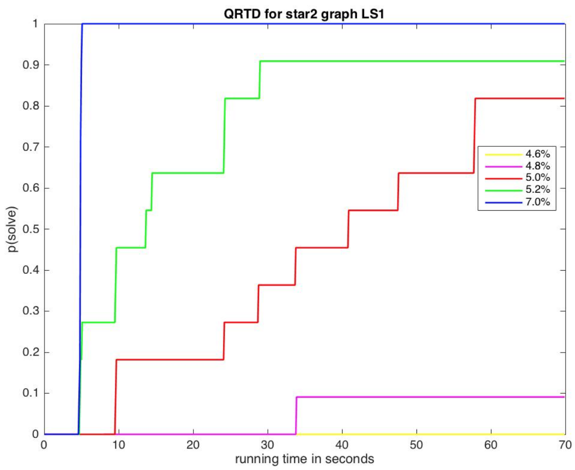

Figure 2: QRTD star2 LS1

5.3.3 Solution Quality Distribution. The figures below show the SQD plots for Simulated Annealing for Hill Climbing.

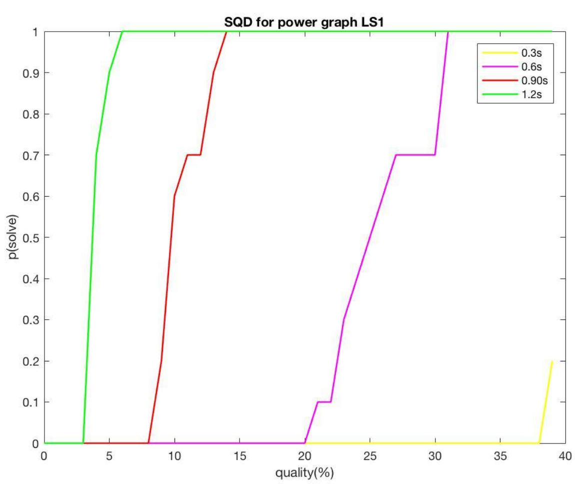

Figure 3: SQD power LS1

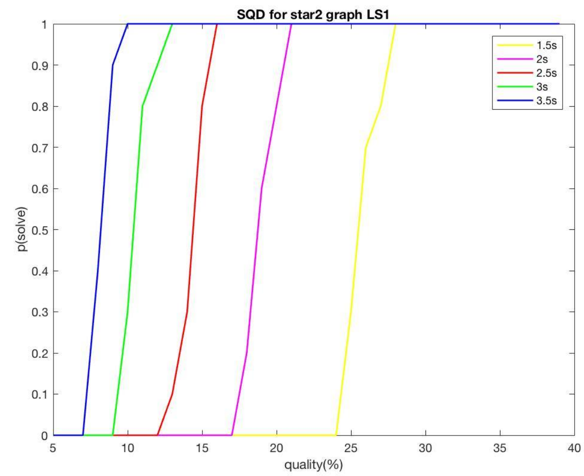

Figure 4: SQD star2 LS1

5.3.4 Box Plots. For **power.graph**, we can see that we can actually reach a good result (5%) in about 1 second. But it will keep running until it reaches about 2.6%.

And the plot for **star2.graph** is even more interesting, it takes awhile to reach 5% error, however it can reach 7% within a very short time.

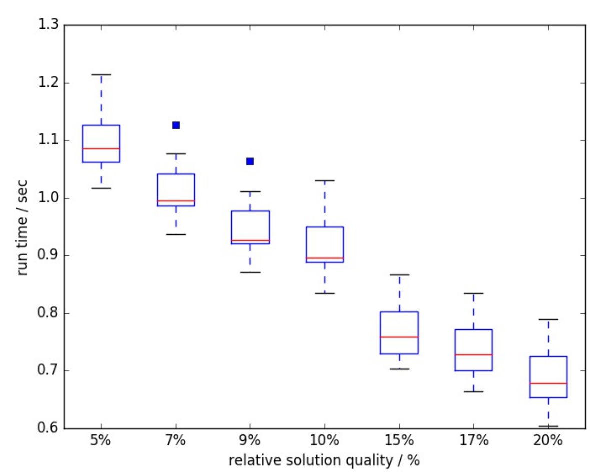

Figure 5: box plot power LS1

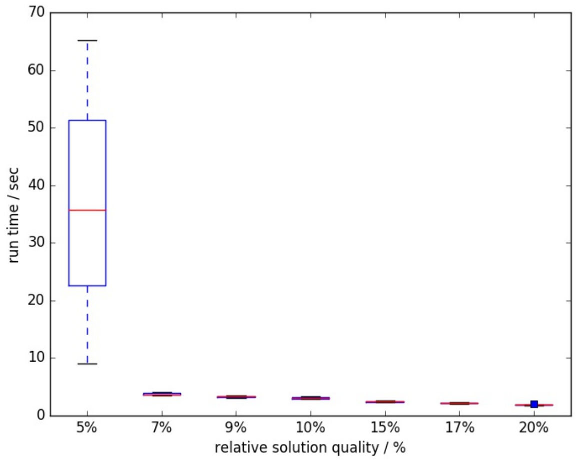

Figure 6: box plot star2 LS1

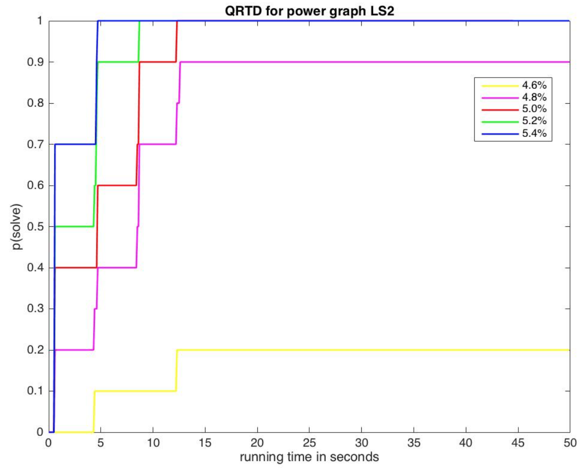

Figure 7: QRTD power LS2

5.4 Local Search - Simulated Annealing

Environment

- Processor: 2.3 GHz Intel Core i5
- Memory: 16GB 2133MHz LPDDR3

5.4.1 Comprehensive Table. Just like Hill climbing algorithm, the comprehensive results are obtained by running 10 different seeds for 10 different runs, and taking the average.

Dataset	Time(s)	MVC	Optimal	RelErr
jazz	4.40	160	158	0.0133
karate	0.0	15	14	0.0071
football	0.51	96	94	0.0213
as-22july06	8.79	3327	3303	0.0073
hep-th	7.72	4035	3926	0.0277
star	10.64	7508	6902	0.0878
star2	6.14	4809	4542	0.0588
netscience	9.62	906	899	0.008
email	4.79	606	594	0.0497
delaunay_n10	5.21	754	703	0.0731
power	6.12	2306	2203	0.0468

5.4.2 Quality Run Time Distribution. From the figures below, we can see that in **power.graph** we can achieve 5.4% error in 5 seconds, and runs to 4.8% within 15 seconds. As for **star2.graph** it's interesting to see that all the qualities converges at about the same time, which means that it only depends on the first update, and the coming runs do not improve the solution.

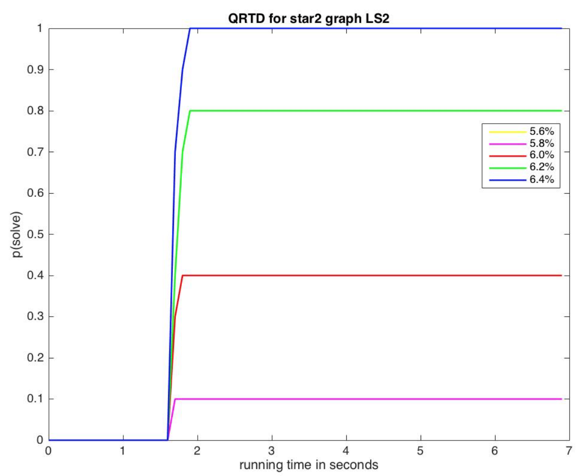

Figure 8: QRTD star2 LS2

5.4.3 Solution Quality Distribution. The figures below show the SQD plots for Simulated Annealing for Simulated Annealing, I will further discuss these plots in the discussion part.

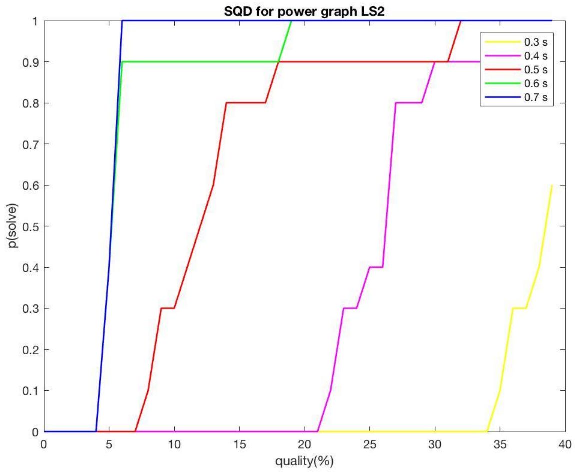

Figure 9: SQD power LS2

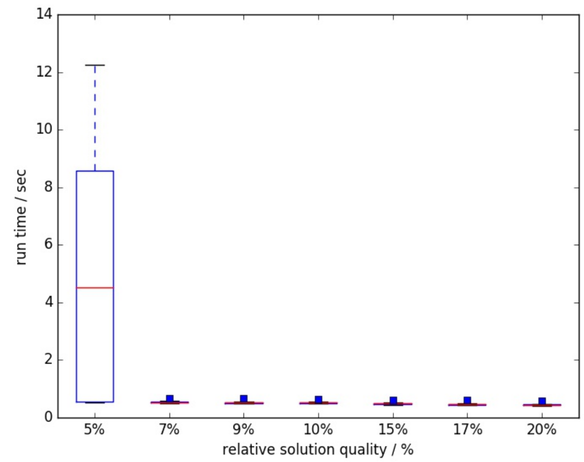

Figure 11: box plot power LS2

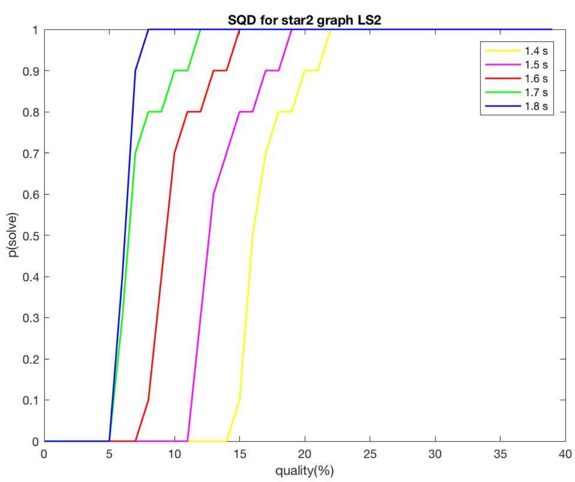

Figure 10: SQD star2 LS2

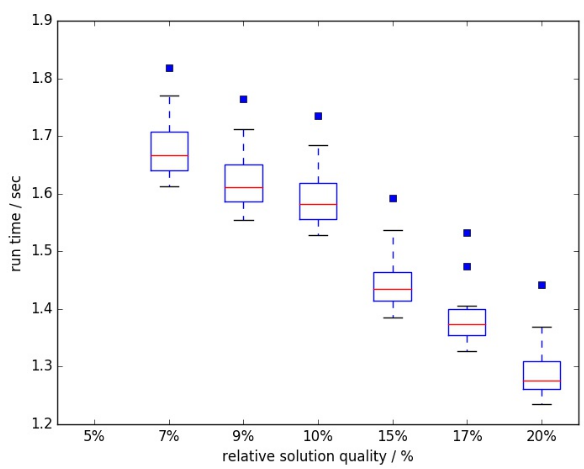

Figure 12: box plot star2 LS2

5.4.4 *Box Plots.* We can see from the box plot **power.graph** that Simulated Annealing soon converges and get 7% error, and takes a few more seconds to get 5% error. The interesting part is that, for the same graph, Hill climbing takes about 30 seconds to reach this (Figure 6).

6 DISCUSSION

Our Branch-and-bound algorithm uses the DFS scheme to try to obtain a feasible solution. This is quite important in some real world situation where we have to get a solution for further usage. Thus, we can see that even with the exponential time complexity, the algorithm still finds promising solutions, some of which are better than some approximation results, for most graphs except the four large graphs.

The approximation algorithm based on DFS has a guaranteed approximation ratio of 2, and out experiment results are all within this approximation ratio. The method presented in the paper does not decide the starting point for groing the spanning tree, therefore, here, we introduce a method which chooses the random starting point given a random seed for each seed, the soltion is deterministic.

Multiple experiments will give us a best starting point to grow this spanning tree.

For the comparison of two local search algorithms, we can look at the comprehensive table, and we can clearly see that hill climbing has a better solution. However, it runs slower than simulated annealing in general. From the plots we can see that for **power.graph**, hill climbing can reach 3% error, and simulated annealing can only get to about 5%, however, it takes about 25 seconds to get there, and simulated annealing only needs 10 seconds. When we see the box plot, hill climbing will need 30 seconds for **star2.graph** to get desired result, and simulated annealing only needs 1.7 seconds. We can also see from the SQD plots that simulated annealing runs faster than hill climbing.

7 CONCLUSION

For this project, four methods are applied to solve minimum vertex cover problem. From our results, we can come up with some guidelines of when to use what kind of algorithm. In general, the local search algorithms performs uniformly better, regardless of the graph size and also only use a short period of time. However, it may not necessary provide an optimal solution even for the small size graphs. Branch-and-bound algorithm, integrated with efficient approximation algorithms, can handle the small size problems quite well and is more likely to give optimal solutions due to its algorithm nature.

REFERENCES

[1] David Avis and Tomokazu Imamura. 2007. A list heuristic for vertex cover. *Operations research letters* 35, 2 (2007), 201–204.

[2] L Sunil Chandran and Fabrizio Grandoni. 2005. Refined memorization for vertex cover. *Inform. Process. Lett.* 93, 3 (2005), 125–131.

[3] Jianer Chen, Iyad A Kanj, and Weijia Jia. 2001. Vertex cover: further observations and further improvements. *Journal of Algorithms* 41, 2 (2001), 280–301.

[4] Jianer Chen, Iyad A Kanj, and Ge Xia. 2010. Improved upper bounds for vertex cover. *Theoretical Computer Science* 411, 40-42 (2010), 3736–3756.

[5] Thomas H Cormen. 2009. *Introduction to algorithms.* MIT press.

[6] François Delbot and Christian Laforest. 2008. A better list heuristic for vertex cover. *Inform. Process. Lett.* 107, 3-4 (2008), 125–127.

[7] François Delbot and Christian Laforest. 2010. Analytical and Experimental Comparison of Six Algorithms for the Vertex Cover Problem. *J. Exp. Algorithmics* 15, Article 1.4 (Nov. 2010), 1.17 pages. DOI: http://dx.doi.org/10.1145/1865970.1865971

[8] Michael R Garey and David S Johnson. 1979. Computers and intractability. A guide to the theory of NP-completeness. A Series of Books in the Mathematical Sciences. (1979).

[9] Carla Savage. 1982. Depth-first search and the vertex cover problem. *Inform. Process. Lett.* 14, 5 (1982), 233–235.

[10] Carsten Witt. 2012. Analysis of an iterated local search algorithm for vertex cover in sparse random graphs. *Theoretical Computer Science* 425 (2012), 117–125.