Minimum Vertex Cover

Shangru Yi
syi73@gatech.edu
Georgia Institute of Technology

Rao Fu
rfu39@gatech.edu
Georgia Institute of Technology

Zhe Zhou
zhezhou@gatech.edu
Georgia Institute of Technology

Huizi Shao
hshao44@gatech.edu
Georgia Institute of Technology

1 INTRODUCTION

The Minimum Vertex Cover (MVC) problem is a typical NP-complete problem that plays an significant role in many practical applications such as operations research, network security, computational biology and parallel machine scheduling. In this paper, four algorithms, Branch-and-Bound, Approximation, Local Search-TW, Local Search-SA would introduce to obtain the minimum vertex cover for the problem sets. Further explanations and discussion to each algorithm would also be presented.

2 PROBLEM DEFINITION

With a set of vertices V and a set of edges E in an undirected graph G = (V, E), a vertex cover is a subset C $\subseteq$ V such that $\forall$ (u, v) $\subseteq$ E: u $\subseteq$ C $\vee$ v $\subseteq$ C. The Minimum Vertex Cover problem is to minimum |C|, which is to minimize the number of the subset of vertices that the algorithm finds.

3 RELATED WORK

Branch-and-bound uses lower bound and upper bound to bound the result while searching. The article [1] shows three different ways of calculating lower bounds of vertex cover problem. We choose the one using max degree because it's simple and gives a relative good result.

Approximation can produce solutions with proven quality. The article [2] compares several approximation algorithms related to the minimization vertex cover problem. The maximum degree greedy algorithm is chosen to implement because of the good performance and low variance.

Local search algorithms for MVC have shown efficiency, with a heuristic idea to assign some scores to nodes or edges. However, single edge-weighting algorithms are not robust, given its greedy search strategy. As for graph instances that defeats greedy heuristics, the performance is not good [3]. Thus, authors in [3] introduce a two-weighting mechanism to counter the greedy effect of single weight. By considering two factors when selecting the next candidate node to move, this mechanism is expected to increase the search space and avoid the typical greedy local traps. We have implemented the ***Two-Weighting (TW)*** algorithm as one of our local searches. On the basis of this algorithm, we also add some restart criteria to allow extremely deviating solutions to be back to previous good state for extra rounds of exploration.

There are few papers directly talking about implementing the simulated annealing algorithm on the minimum vertex cover problem. However this is due to the fact that the simulated annealing algorithm is a classic algorithm that is well studied. The paper [4] talks about an efficient simulated annealing algorithm for the minimum vertex cover.

4 ALGORITHMS

Four algorithms, Branch-and-Bound, Approximation, Local Search-Two Weighting, Local Search-Simulated Annealing were implemented and we have detailed descriptions for each as follows.

4.1 Branch-and-Bound

4.1.1 *Description.*

Branch-and-Bound is a popular algorithm when we have to deal with the optimization problem. Branch-and-Bound is used to find optimal solution with a search Tree and decision bounds: both lower bound and global upper bound.

First, we sort the list of vertices in terms of degree for that higher degree is more promising in including in vertex cover. Next, we start with the vertex with max degree and expand the search tree with two branches, which are including the vertex and excluding the vertex. Next, we recalculate the lower bound of each instance by using sum of edges divided by max vertex degree in the remaining vertex set.

There are generally three possible output after expanding. First,if we have reach the end of our problem and find a solution, we can update the upper bound. Second, we recalculate the lower bound and we find that it's greater than the upper bound, we prune it. The last category would be we the results are valid and we continuously expand the next level of branches.

4.1.2 *Pseudo-Code.*

The Pseudo-Code is shown in *Algorithm 1*

4.1.3 *Analysis.*

The branch-and-bound algorithm takes exponential time. In our

CSE6140 Computational Science & Engineer Algorithms, Fall 2020, Atlanta, GA

Algorithm 1: Branch and Bound Pseudo-Code

```
Data: Graph with Vertexes V and Edges E
Result: List of Vertex(Vertex Cover)
Vertex Cover <- ∅
OptVC <- ∅
UB <- V //Upper Bound
while VC not empty do
    select a vertex u of maximum degree from V and expand
    V <- V - u;
    E <- E - e;
    cost(e);
    for Config in the new added configuration do
        if solution found then
            update UB;
        else if LB+cost(e) ≥ UB then
            prune;
        else
            Vertex Cover <- VC ∪ e
return OptVC;
```

method, for each vertex, we expand on the exist vertex cover to include or exclude the vertex. We traverse each vertex to the end, which means all vertex is visited, if we meets a solution or the lower bound calculated is greater than the upper bound, we backtrack our search tree.Every time if we find a new optimal vertex cover, we take record of the lowest one.

The method guarantee to find the optimal solution if these is not specific time limits, and because of using backtracking and search tree, the running time takes exponential time.

4.2 Approximation

4.2.1 Description.
The Maximum Degree Greedy Algorithm came from the known greedy algorithm for solving the set cover problem. Firstly, the algorithm sorts all nodes in the graph in descending order by degree of each node. After that, the algorithm continuously removes the max degree node and its corresponding edges, and in the meantime, adds that node in the vertex cover, until there are no edges left in the graph. In this way, the vertex cover is obtained that covers all edges in the graph.

4.2.2 Pseudo-Code.
The Pseudo-Code is shown in *Algorithm 2*

Algorithm 2: Approximation Pseudo-Code

```
Data: Graph with Vertexes V and Edges E
Result: List of Vertex(Vertex Cover)
Vertex Cover <- ∅
while E > 0 do
    select a vertex u of maximum degree from V
    e is the corresponding edges of u
    V <- V - u;
    E <- E - e;
    Vertex Cover <- C ∪ u;
return Vertex Cover
```

4.2.3 Analysis.
The algorithm sorts the nodes based on the degree of each node which consumes O(vlogv) time complexity. For each max degree node in each iteration, it removes the node and its corresponding edges which consume O(v * (e/v)) where e/v is the average edges of each node. Overall, time complexity of the algorithm is O(v * (e/v)). The algorithm needs a graph which is a dictionary where the key is the node and the value is the set of adjacent node values. This is O(v*(e/v)). It also maintains an array to store the sorted node with decreasing node degree. Then this is O(v). Therefore, the Space complexity is of O(v * (e/v)) where e/v is the average adjacent node of each node.

The method is guaranteed 2 approximate, which is smaller than the number of edges selected. The worst case is H(δ) where H(n) is 1 + ½ + … + 1/n and δ is the maximum degree of the graph. The average behavior of the algorithm is proven in [2] and moreover, when n tends to be infinity, the limit of the expected approximation ratio is $1 + e^{-2}$.

The approximation algorithm using the greedy method is easy to understand and implement. Furthermore, it is a really fast algorithm with the worst case running time polynomial and a bound of the quality of the solution. However, the optimal solution can't be guaranteed for this method.

4.3 Local Search - TW

4.3.1 Description.
The basic idea of this algorithm is to employ edge weighting techniques and prefer to select vertex with higher weighted score, which is also widely adopted by lots of previous efforts for MVC problem.

To avoid failures in typical greedy solutions, a penalty is introduced for the those vertices if they are not in the current candidate solution C, and those vertices are forced to be selected into candidate C[3].

Each edge $e \in \mathcal{E}$ is associated with a non-negative integer number $w(e)$ as its weight. Thus, the cost of a candidate solution is defined as the sum of uncovered edges. As for each vertex, there are two states as indications of whether the vertex is included in the candidate solution. Thus, the benefit of changing the state of a vertex would be $cost(C) - cost(C')$, where $C' = C \backslash \{v\}$ if $v \in C$ and $C' = C \cup \{v\}$ otherwise. The benefit is treated as the *score* of a vertex. The cost of a solution C, i.e. $cost(C)$, is the weight sum of its uncovered edges.

With similar idea as *Tabu* search, a notion of ***configuration change*** is introduced in the algorithm. A vertex is said to be configuration changed, if and only if at least one of its neighbors has changed

their states since the last removal of that vertex from searching solution.

Restart mechanism has been added to the algorithm to prevent extremely deviating solution. *restart* controls whether to restore current solution to previous searched best. After removal of vertex with greatest *score*, we check the number of edges that are uncovered. If the number is greater than *epsilon*, we treat this solution as a deviating one. If the number is too large, i.e. greater than α, or *restart* has reached *threshold*, we reverse current solution and restore both edge weights and vertex weights.

The parameters from original algorithm are also preserved. We leave the details to readers, but illustrate them briefly. *gamma* controls the cycle to periodically reduce the edge weights to avoid the increasing weight of frequently uncovered edges. δ cooperates with vertex weights to counter the single edge weighting, and β prevents vertex weights from always increasing.

4.3.2 Pseudo-Code.

The Pseudo-Code is shown in *Algorithm 3, 4*

4.3.3 Analysis.

Generally, the introduced vertex weighting brings more possibility for those unexpected vertexes to be added to the searching solution, and thus avoid the greedy local optimal. Also, the decreasing mechanism for vertex weights would prevent an expected vertex to be selected too often due to its high vertex weight [3].

During the execution of original Two-Weighting without adding restart mechanism, we notice that the there is a bad increasing trend for the size of uncovered edges as the algorithm running. It will remain the same or take long for the size to decrease. Thus, we tuned several thresholds to indicate the possible needs for restarts. Once restarted, we reverse current search solution to the previous searched vertex cover and reinitialize the weights for nodes and edges. This restart mechanism allow the revised version search more possible space under the good states, instead of just an one-time random dice tossing. It is noticeable that the solution quality is much better and the vertex cover rate is also higher.

Algorithm 3: Two-Weighting Local Search Pseudo-Code

```
Data: graph G = (V, E)
Result: vertex cover of G
step <- 0;
restart <- 0;
initialize edge weights w(e), and node weights w_v;
C <- approximateVC(G);
while (elapsed time < cutoff) and (size(C*) > opt) do
    if C is vertex cover then
        C* <- C;
        select a vertex with highest score from C;
        continue;

    select a vertex u ∈ C with highest score from C;
    C <- C\{u};
    if |uncovered edges| >= ε then
        if |uncovered edges| >= α or restart ≥ threshold
          then
            C <- C*;
            reinitialize edge weights and vertex weights;
              continue;
        restart++;

    randomly choose an uncovered edge e;
    v <- chooseVertex(e);
    C <- C ∪{v};
    foreach uncovered edge e do
        w(e) + +;

    if step %γ = 0 then
        foreach edge ∈ E do
            w(e) − −
    foreach vertex v ∉ C do
        w_v(v) + +;
    if step %100 = 0 then
        foreach vertex v with w_v(v) > 1 do
            w(e) − −

    step++;
return C*;
```

4.4 Local Search - SA

4.4.1 Description.

The Simulated Annealing algorithm is a well studied local search algorithm. It involves stochastic decision to avoid local maxima. Applying to the minimum vertex cover problem, we use the approximation result as the initial solution. We expand the solution space by randomly removing vertices in the solution and we randomly select uncovered edges and choose one of the two side vertices and add to the solution set. If compared to the old solution, moving and adding two different vertices gives us a less amount of uncovered edges (i.e. scores), then we say it's a better solution. If it's not a better solution, then we use the temperature to choose whether to accept the worse solution or not.

4.4.2 Pseudo-Code.

The Pseudo-Code is shown in *Algorithm 5*

4.4.3 Analysis.

The Simulated Annealing (SA) algorithm starts with the initial solution. It then loops for some number of iterations. For every iteration it randomly removes vertices from the vertex cover until there are some uncovered edges. Then we take a snapshot of the old vertex cover. Next we randomly replace one vertex in the vertex cover with one of the uncovered vertex. We compare the number of edges being covered between the new solution and the old solution. If

Algorithm 4: chooseVertex (TW) Pseudo-Code

```
Data: an uncovered edge e
Result: a vertex (one of endpoints) v
if only one endpoint is configuration changed then
    return v ∈ e that v is configuration changed;

if |w_v(v_1) − w_v(v_2)| > δ then
    v* = v ∈ e that v has greater vertex weight;
    w_v(v*)* = β;
    return v*;
else
    return v has greater score;
return v;
```

Algorithm 5: Simulated Annealing Pseudo-Code

```
Data: Graph with Vertexes V and Edges E
Result: List of Vertex(Vertex Cover)
Vertex Cover <- approximation(V,E)
Global Best Cover <- Vertex Cover while i < max steps do
    while j < some fixed steps do
        Randomly select v_1 to remove from Vertex Cover
        Randomly select v_2 to add to Vertex Cover
        if new solution covers more edges then
            keep the new solution
        else
            with probability T, accept the worse solution
    T <- T · α
return Vertex Cover
```

the new solution is better, it keeps the new solution. If it's worse, then with some probability we are going to keep the worse solution.

There is no guarantee that SA can help us find the global optimal. But it can usually do better than the local optima.

5 EMPIRICAL EVALUATION

5.1 Platform

- OS: Windows 10
- CPU: 2.60 GHz Intel Core i7-9750H
- Memory: 32 GB 2667 MHz DDR4
- Language: Python 3.7.7

5.2 Branch-and-Bound

5.2.1 Experimental Procedure and Evaluation Criteria.
The algorithm is implemented from the branch-and-bound algorithm with binary decision we talked in class. The lower bound calculate follows the degree based lower bound calculation from[1].The evaluation metrics can be the vertex cover value we generated, compared to the optimal value provided and the running time of the algorithm.

5.2.2 Results.
The table below is the example results of our Branch and Bound algorithm.

Table 1: Branch and Bound

Dataset	Time(s)	VC Value	OPT	ReErr
as-22july06	224.350	3304.000	3303.000	0.000
delaunay_n10	4.770	739.000	703.000	0.051
email	1.030	604.000	594.000	0.018
football	0.580	94.000	94.000	0.000
hep-th	85.200	3947.000	3926.000	0.005
jazz	306.250	158.000	158.000	0.000
karate	0.000	14.000	14.000	0.000
netscience	2.240	899.000	899.000	0.000
power	24.690	2212.000	2203.000	0.004
star	183.000	7366.000	6902.000	0.067
star2	169.040	4677.000	4542.000	0.030

5.3 Approximation

5.3.1 Experimental Procedure and Evaluation Criteria.
The algorithm is implemented based on one of the methods in *Analytical and experimental comparison of six algorithms for the vertex cover problem*, which is called Maximum Degree Greedy Algorithm. The code followed the logic described in the paper but has problems with performance when encountering a bigger dataset. The evaluation metrics can be the vertex cover value we have, compared to the optimal value provided and the running time of the algorithm.

5.3.2 Results.
The table 2 is the example results of Approximation algorithm.

Table 2: Approximation

Dataset	Time(s)	VC Value	OPT	ReErr
as-22july06	10.550	3307.000	3303.000	0.001
delaunay_n10	0.170	737.000	703.000	0.048
email	0.170	605.000	594.000	0.019
football	0.090	96.000	94.000	0.021
hep-th	3.800	3944.000	3926.000	0.005
jazz	0.100	159.000	158.000	0.006
karate	0.100	14.000	14.000	0.000
netscience	0.200	899.000	899.000	0.000
power	1.420	2277.000	2203.000	0.034
star	8.340	7374.000	6902.000	0.068
star2	9.410	4697.000	4542.000	0.034

5.4 Local Search - TW

5.4.1 Experimental Procedure and Evaluation Criteria.
The Two-Weighting algorithm uses the result from approximation as the initial solution. The optimal value for the minimum vertex cover size is passed for cutoffs. If current vertex cover size is the same as the optimal, the algorithm is terminated.

Table 3: Two Weighting

Dataset	Time(s)	VC Value	OPT	ReErr
as-22july06	41.180	3303.000	3303.000	0.000
delaunay_n10	33.370	703.000	703.000	0.000
email	3.590	594.000	594.000	0.000
football	0.100	94.000	94.000	0.000
hep-th	392.840	3927.000	3926.000	0.000
jazz	0.100	158.000	158.000	0.000
karate	0.100	14.000	14.000	0.000
netscience	0.200	899.000	899.000	0.000
power	142.580	2203.000	2203.000	0.000
star	528.390	6911.000	6902.000	0.001
star2	475.210	4543.000	4542.000	0.000

Table 4: TW Box plot stats for power.graph

Relative Error (%)	lower quartile	median	upper quartile
0	61.7175	93.670	128.8100
0.1	29.3075	39.275	47.4500
0.1	20.9075	25.475	30.0025
0.2	10.4550	12.585	20.9250
0.4	9.3950	9.755	10.2475

We set the parameters as following: $\gamma = max(totaldegree//8, 1)$, $\delta = 10000$, $\beta = 0.8$, $\epsilon = 5$, $\alpha = 50$, and $threshold = 0.3 * \delta$.

5.4.2 Results.
The table 3 is the example results (random seed 123) of Two-Weighting algorithm.

5.4.3 Evaluation Plots.
We select 50 random seeds for experiment repeats (117 - 215, with 2 as gap). For **QRTDs**, we set the relative error as 0%, 0.1%, 0.3%, 0.6% and 1% for **power**, and 0%, 0.1%, 0.5%, 0.8% and 1% for **star2**. For cutoff time in **SQDs**, we set the time as 6, 8, 10, 16, 24, 40 and 60 for **power**, and 30, 40, 50, 100, 150, 200, and 300 for **star2**. The relative error for initial solution is around 3% for both **power** and **star2**. It is clear that Two-Weighting can reach good quality within 10s. We can also reach similar conclusion from **SQDs**. As for boxplot for running time, it is clear that as the relative error increase, the data tends to become more compact, indicating that randomization does not affect the solution quality.

The Qualified Run time plots for Local Search 1-Two weighted is shown in **figure 1**. The Solution Quality Distribution plots for Local Search 1-Two weighted is shown in **figure 2**. The Run time box plots is shown in **figure 3**, which provide more information. Furthermore, the box-plots related data is shown in **table 4** and **table 5**.

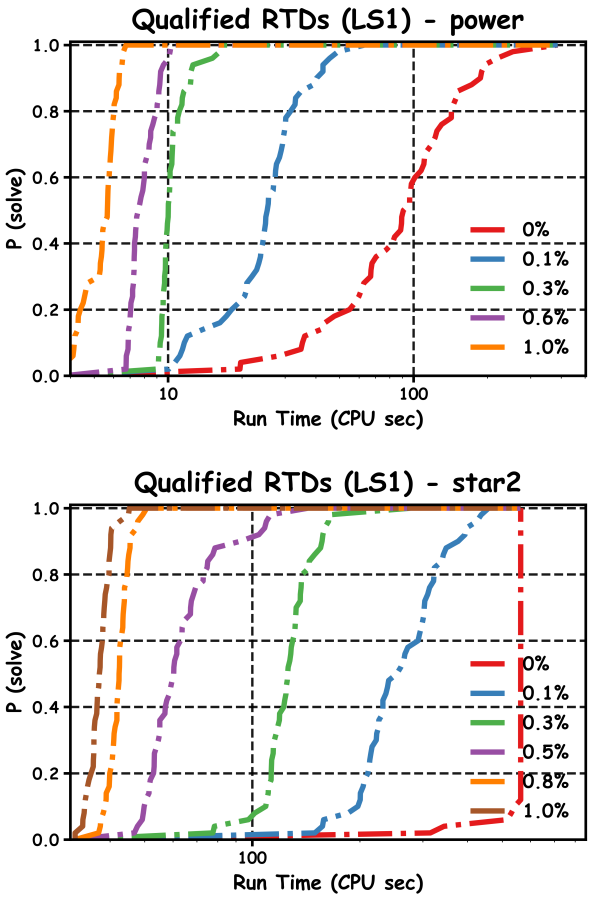


Figure 1: Qualified Runtime plot for LS1: TW

Table 5: TW Box plot stats for star2.graph

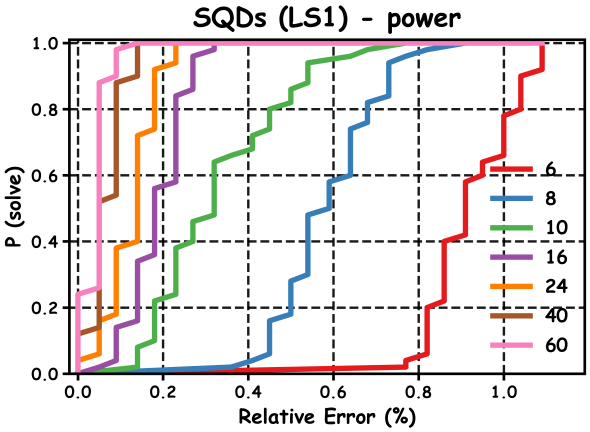
Relative Error (%)	lower quartile	median	upper quartile
0	386.9250	523.195	538.9550
0.1	281.8400	331.135	414.1700
0.1	215.0375	256.845	312.2300
0.2	138.5725	149.120	182.7975
0.4	71.2325	104.875	113.6850
0.8	40.7675	42.465	43.9175

5.5 Local Search - SA

5.5.1 Experimental Procedure and Evaluation Criteria.
The Simulated Annealing algorithm uses the result from approximation as the initial solution. We compare the result of the Simulated Annealing with original approximation result. And we calculate the relative percentage of improvement.

We start with Temperature = 0.6 and cooling factor = 0.95. We perform 70 rounds of cooling over the 30000 searching iterations.

5.5.2 Results.
The table 6 is the example results of the Simulated Annealing algorithm.



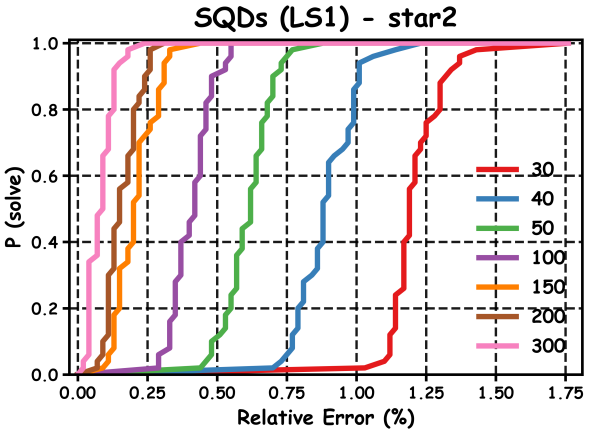
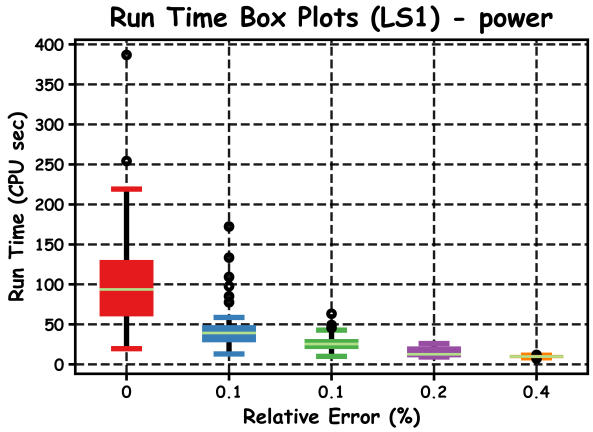


Figure 2: Solution Quality Distributions plot for LS1: TW



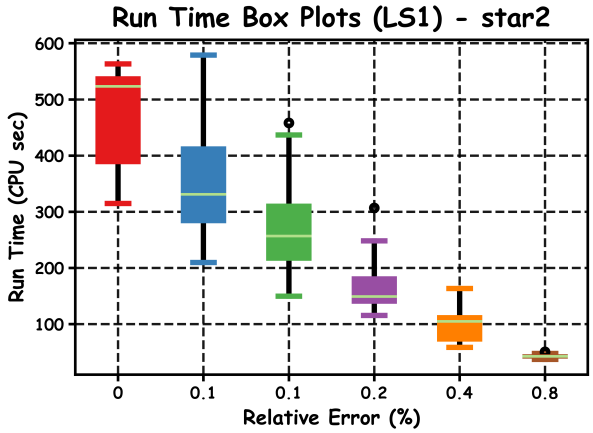


Figure 3: Run Time Box Plot for LS1: TW

Table 6: Simulated Annealing

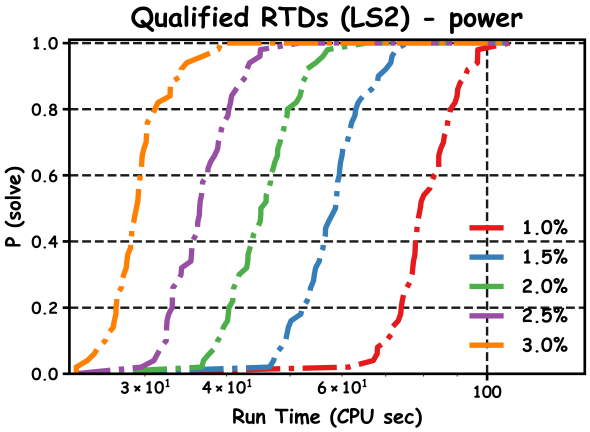
Dataset	Time(s)	VC Value	OPT	ReErr
as-22july06	172.870	3303.000	3303.000	0.000
delaunay_n10	35.930	707.000	703.000	0.006
email	29.730	594.000	594.000	0.000
football	5.960	94.000	94.000	0.000
hep-th	198.480	3930.000	3926.000	0.001
jazz	8.950	158.000	158.000	0.000
karate	2.100	14.000	14.000	0.000
netscience	42.820	899.000	899.000	0.000
power	117.630	2212.000	2203.000	0.004
star	392.160	7313.000	6902.000	0.059
star2	251.210	4597.000	4542.000	0.012

5.5.3 Evaluation Plots.
We select 50 random seeds for experiment repeats (117 - 215, with 2 as gap) the same as the setting for Two-Weighting. Since Simulating Annealing does not reach optimal, we adopt difference setting for relative error and cutoff time. For **QRTDs**, we set the relative error as 0%, 0.1%, 0.3%, 0.6% and 1% for **power**, and 0%, 0.1%, 0.5%, 0.8% and 1% for **star2** which both can be found in **figure 4**. For cutoff time in **SQDs** which can be found in **figure 5**, we set the time as 40, 50, 60, 70, 80, 90 and 100 for **power** , and 80, 100, 125, 150, 175, 200 and 225 for **star2**. The relative error for initial solution is around 3% for both **power** and **star2**. From boxplot, the whisker indicates the randomization would affect the solution quality more, comparing with Two-Weighting.

The Run time box plots is shown in **figure 6**, which provide more information. Furthermore, the box-plots related data is shown in **table 7** and **table 8**.

Table 7: SA Box plot stats for power.graph

Relative Error (%)	lower quartile	median	upper quartile
0.5	100.8200	105.530	105.920
1.0	75.6300	79.180	87.000
1.6	50.1275	54.205	59.925
2.3	37.3475	40.305	42.865
3.0	27.6325	29.180	30.250



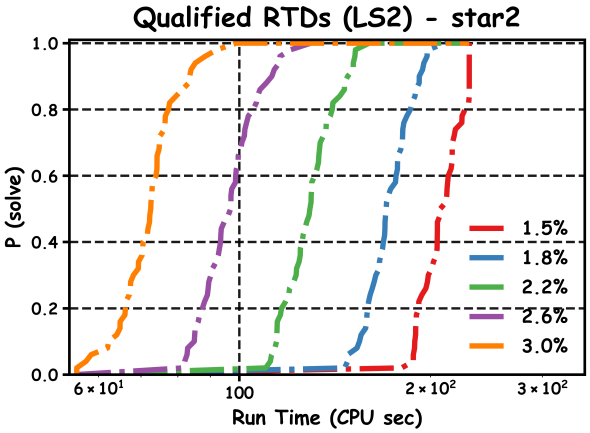
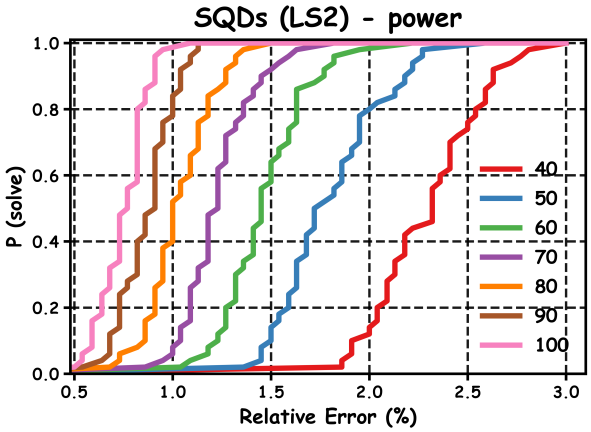


Figure 4: Qualified Runtime plot of LS2-SA



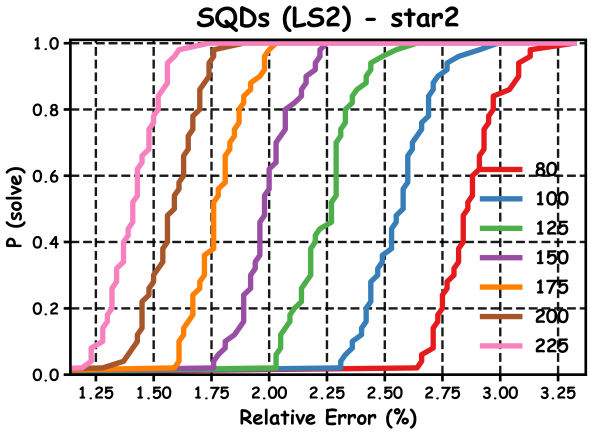


Figure 5: SQD plot for LS2-SA

Table 8: SA Box plot stats for star2.graph

Relative Error (%)	lower quartile	median	upper quartile
1.5	192.2925	205.100	216.0950
1.8	162.2550	170.495	181.4800
2.2	120.4600	129.460	137.0775
2.6	89.3825	96.965	102.5225
3.0	67.3875	72.560	76.3500

6 DISCUSSION

The Branch-and-Bound algorithm is expected to take longer time to do the exhaust search. Because in the algorithm, we have first sorted the vertices with the max degree, the first solution was found quickly. After the first solution was found, it took larger time to find the second and more optimal solutions. We have presented the result on table Branch and Bound. Within the limit time of 600 s, we found 4 datasets that reached the optimal vertex cover. Given the enough time, the Branch and Bound should reach the optimal solution.

To improve the algorithm, we first use the result of approximation as the starting point of the algorithms. Because we decided the lower bound in terms of degree, which might be a looser bound, the running time to reach the optimal result can take longer. To further improve the algorithm, we can research and try different lower bound.

The Approximation algorithm is viewed as the base algorithm for the implementation process. We used the greedy algorithm by sorting the edges for each nodes and removing from the maximum degree of the node and edges. The algorithm has simple logic and can finish running quickly. There is no limit set for the algorithm, since all the cases can be finish in reasonable time. The maximum time taken for the example cases is around 11s, which is **as-22july06**. The relative error is low for the maximum greedy as well, with the worst case 0.068 for star.graph. Overall, it gives acceptable performance and good running time.

The Simulated Annealing local search algorithm requires a careful selection over the initial temperature T and the cooling factor α. Another interesting factor is the minimum number of iteration steps

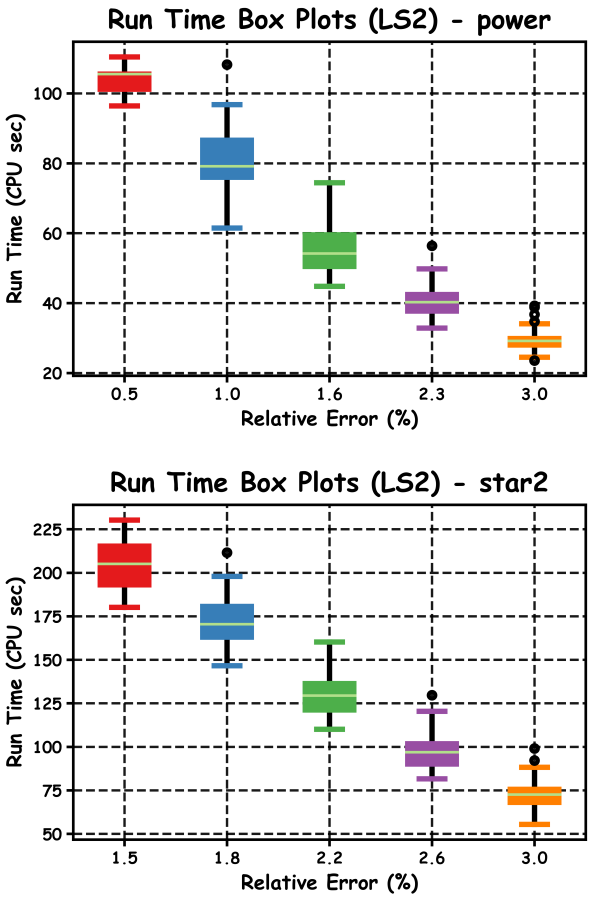


Figure 6: Run Time Box Plot for LS2: SA

before the temperature can change. We call this *steps*. A larger T yields a broader search space and given enough time it will likely give a better solution compared to a lower T. But it also takes longer time to converge. $\alpha = 0.95$ seems to be an effective cooling factor after some experiments. And I take $steps \approx 300$ given 20000 total iterations. If there are more iterations then the *steps* needs to adjust accordingly in order not to cool down too quickly. As a result the Simulated Annealing achieves decent accuracy on most testing files while it takes a longer time to yield the results.

It is easy to tell from **figure 7** that Two Weighting Local search yield stable and good results. Comparing with simulated annealing, the box plot for the vertex cover size with cutoff time as 600s indicates small fluctuations in quality. This would be credited to the guided weighting mechanism. For Simulating Annealing, the only guidance would be the degree information, while both node weights and edge weights are incorporated to the selection progress for Two Weighting.

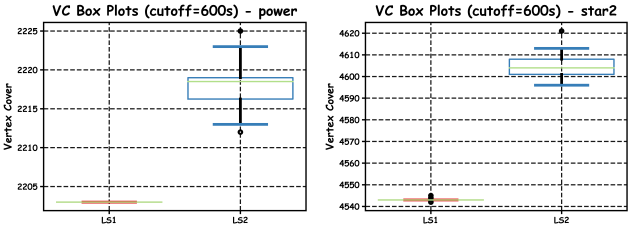


Figure 7: VC Size Box Plot for Local Search

6.1 Structure Visualizations

We perform sampling to the nodes in graphs and generate the structure visualization for several graphs in the dataset in **figure 8**. We color the node based on the number of its edges. Blue indicates very few edges and red indicates lot of neighboring nodes, while other colors stay between.

We start from some small graph, i.e. **email** and **delaunay_n10**. It is easy to tell that the approximation algorithm and local search algorithm both show low relative error. It is more or less due to the relative clear level structure. The nodes colored as not blue is clear a good candidate for vertex cover. Also, this clear graph structure would decrease the possible search space, due to the great gap between node scores in local search (score is based on the number of neighboring nodes or degree).

As for those big graph, i.e. **star** and **as-22july06**, it would take longer to search for possible good candidates. Also, with basically all nodes colored in blue, those nodes will be treated equally and no one except those in red and yellow would stand out. The possible search space is much larger and it is harder to find good vertex cover.

7 CONCLUSION

The four algorithms, Branch-and-Bound, Maximum Degree Greed Approximation, Two Weighting Local Search and Simulated Annealing Local Search, are detailed and implemented in the paper. The testing results show all four algorithms have pros in some aspects and cons in other aspects. The Maximum Degree Greed Approximation is fast yet effective, which yields a quite accurate solution using most of the test files. The other three algorithms use the result of the approximation as an initial solution. The Branch-and-Bound algorithm takes a lot of time to exhaustively explore prospective search spaces. The Simulated Annealing algorithm requires careful choices of the parameters and is highly stochastic. The two weighting local search is a cutting-edge algorithm that in a fixed time yields a better accuracy than the BnB and the Simulated Annealing.

REFERENCES

[1] Mingyang Li Luzhi Wang, Shuli Hu and Junping Zhou. Analytical and experimental comparison of six algorithms for the vertex cover problem. *MDPI*, 07 2019.
[2] Francois Delbot and Christian Laforest. Analytical and experimental comparison of six algorithms for the vertex cover problem. *ACM Journal of Experimental Algorithms - JEA*, 15, 03 2010.

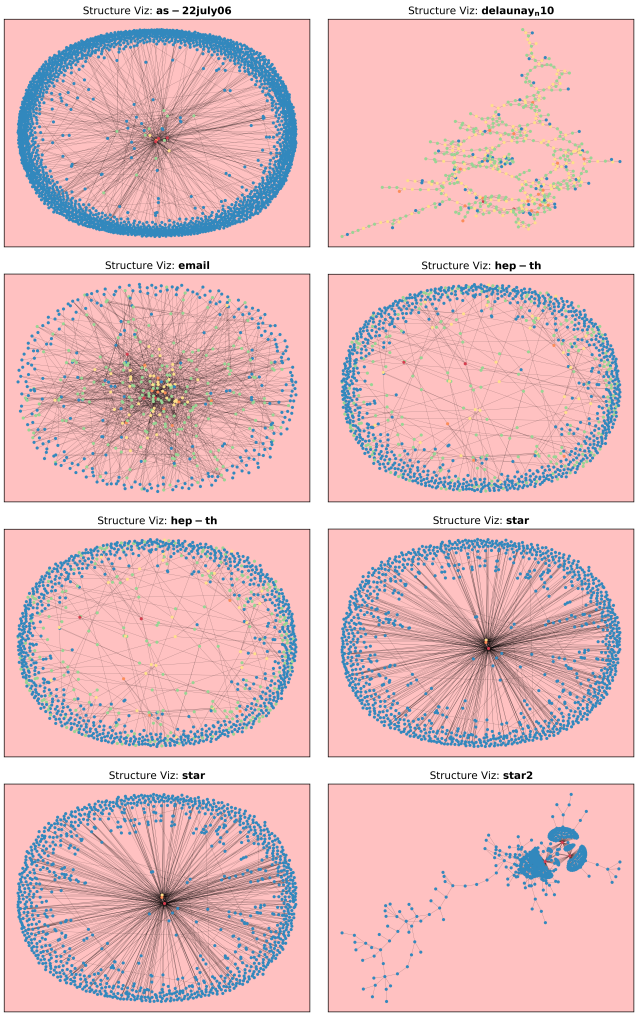


Figure 8: Graph Structure Visualization

[3] Shaowei Cai, Jinkun Lin, and Kaile Su. Two weighting local search for minimum vertex cover. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 1107–1113, 2015.

[4] Xinshun Xu and Jun Ma. An efficient simulated annealing algorithm for the minimum vertex cover problem. *Neurocomputing*, 69(7):913 – 916, 2006. New Issues in Neurocomputing: 13th European Symposium on Artificial Neural Networks.