CSE-6140 Fall2015 Project Report

Shang-Tse Chen
schen351@gatech.edu

Chengwei Li
licw@gatech.edu

Ruiming Lu
rlu39@gatech.edu

Lixi Zhao
zhaolixi@gatech.edu

1. INTRODUCTION

A vertex cover of an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ such that if $(u, v) \in E$, then $u \in V'$ or $v \in V'$ (or both). That is, each vertex "covers" its incident edges, and a vertex cover for G is a set of vertices that covers all the edges in E. The size of a vetex cover is the number of vertices in it. The Minimum Vertex Cover problem is therefore simply to find a vertex cover of minimum size in a give graph. As an example, Fig. 1 shows a graph with 7 vertices and 8 edges, and the light-shaded vertex set $\{b, d, e\}$ indicates a minimum vertex cover.

Since minimum vertex cover problem is a well-known NP-complete problem, we don't expect to find a polynomial-time algorithm for finding a minimum vertex cover. However, because this probelm has numerous application in computational biology, operations research, the routing and management of resources, many methods have been proposed to solve this problem.

In this project, we will implement algorithms that fall into three categoris:

1. Exact algorithm using Branch-and-Bound
2. Heuristics with approximation guarantees
   - Edge Deletion (2-approximation)
   - Maximum Degree Greedy ($\log n$-approximation)
3. Local search
   - Efficient Simulated Annealing [4]
   - Two Stage Exchange with Edge Weight Forgetting [2]

Following the implementation, we conduct empirical analysis of algorithm performance on datasets provided, and discuss the trade-offs between accuracy, speed, etc., across different algorithms.

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

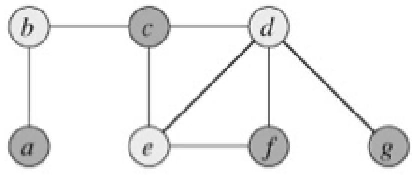

Figure 1: A simple graph showing the minimum vertex cover problem

2. PROBELM DEFINITION

Given a graph $G = (V, E)$, a vertex cover is a set of vertices such that each edge in the graph is incident to at least one of vertex in the set, thus all edges are "covered" by that set of vertices. The set of all vertices, V, is one valid example of vertex cover. As the name suggests, the minimum vertex cover is the problem of finding a vertex cover with minimum cardinality, or in other word, the vertex cover with minimum size. This problem is the special case of the minimum set cover problem and is also NP-Complete (no known polynomial solution, unless $P = NP$). As shown in Fig. 1, the set $V = \{b, d, e\}$ is a minimum vertex cover.

3. RELATED WORK

Minimum Vertex Cover is first proved to be NP-complete by Karp [5]. Therefore, most effort has been trying to get an approximate solution. The simple greedy algorithm can lead to an approximation ration as bad as $\log n$ [6]. However, there are several algorithms with approximation ratio of 2 [1, 7, 8, 9]. Delbot and Laforest (2010) [10] analyze the average-case instead of worst-case performance for 6 2-approximation algorithms and conduct extensive experiments on common graphs. Experimental studies on heuristic approaches on the vertex cover problem include [11, 12, 13].

Whether there exists a better than 2-factor approximation for the minimum vertex cover problem is one of the major open problems in research area of approximation algorithm. It is conjectured that it is impossible to achieve an approximation ration lower than 2 [14]. In fact, it has been proved that there does not exist a better than 1.1666-approximation algorithm for this problem.

In this report, we empirically compare the exact algorithm by branch-and-bound [1], two approximation algorithms described in [10], and two local search algorithms [3, 4].

4. ALGORITHMS

4.1 Branch-and-Bound

In the class we discussed the Branch-and-Bound algorithm to get the exact solution. The key point for Branch-and-Bound method is to calculate the lower bound. Assume that C' is a partical vertex cover of graph G, and we create another new graph $G'(V', E')$ which is subgraph of G without cover by C'. Then the lower bound for this problem is $|C'|$+(lower bound on the VC for G'). We also know that the approx solution quality A is less and equal to 2OPT, then we could get the lower bond for G' is $\frac{1}{2}A$. The details of how to implement the approx algorithm shown on the second part, and here is the algorithm for Branch-and-Bound.

The details of implementation could have 2 methods: first is recursion and second is iteration with stack.

Algorithm 1: Branch-and-Bound Algorithm

```
Input  : graph $G = (V, E)$, the $cutoff$ time
Output: vertex cover of $G$
1 $C' := \emptyset$;
2 $E' := \emptyset$;
3 Intial state calculate the UP (upper bound) of G by
  using approx algorithm while $E' \neq \emptyset$ do
4     if All edges are coverd then
5         if Size $C' <$ UP then
6             Update the result
7         end
8         return
9     end
10    if Size $C' >$ UP then
11        return
12    end
13    Case 1: Choose a vertex $x$ of $G$ and add into $C$‘
      Calculating $G'$;
14    Calculate the LB by call the approx algorithm;
15    if LB > UB then
16        return
17    end
18    Expand by choose the vertex in $V'$;
19    for item in $V'$ do
20        Call backtracking for $V$‘;
21        if found the result then
22            update the upper bound
23        end
24    end
25    Case 2: Do not choose a vertex of $G$, and add all of
      $x$‘s neihbor into $C'$ Calulating $G$‘ Calculate the LB
      by call the approx algorithm;
26    if LB > UB then
27        return
28    end
29    Expand by choose the vertex in $V'$;
30    for item in $V'$ do
31        Call backtracking for $V$‘;
32        if found the result then
33            update the upper bound
34        end
35    end
36 end
37 return $C'$
```

4.2 Approximate algorithm

Algorithm 2: Edge Deletion (ED)

```
Input  : graph $G = (V, E)$
Output: vertex cover of $G$
1 $C := \emptyset$;
2 $E' := E$;
3 while $E' \neq \emptyset$ do
4     Choose an arbitrary edge $(u, v) \in E'$;
5     $C \leftarrow C \cup \{u, v\}$;
6     remove from $E'$ every edge incident on either $u$ or $v$;
7 end
8 return $C$
```

Algorithm 3: Maximum Degree Greedy (MDG)

```
Input  : graph $G = (V, E)$
Output: vertex cover of $G$
1 $C := \emptyset$;
2 $E' := E$;
3 while $E' \neq \emptyset$ do
4     Select a vertex u of maximum degree;
5     $C \leftarrow C \cup \{u\}$;
6     remove from $E'$ every edge incident on $u$;
7 end
8 return $C$
```

We implemented two approximation algorithms. The first algorithm, Edge Deletion (ED), is shown in Algorithm 2. Line 1 initializes C to the empty set. Line 2 sets E' to be a copy of the edge set $G.E$ of the graph. The loop of lines 3-7 repeatedly picks an edge (u, v) from E', adds its endpoints u and v to C, and deletes all edges in E' that are covered by either u or v. Finally, line 8 returns the vertex cover C . The running time of this algorithm is $\mathcal{O}(V + E)$, using adjacency lists to represent E'. This approximation algorithm is a polynomial-time 2-approximation algorithm.

The second algorithm, Maximum Degree Greedy (MDG), is shown in Algorithm 3. In each iteration, instead of choosing an arbitrary edge, it chooses a node u of maximum degree. It then adds u to the vertex cover and removes all edges incident to u. This algorithm has an approximation ratio of $H(\Delta) = 1 + \frac{1}{2} + \cdots + \frac{1}{\Delta}$, where Δ is the maximum degree in the graph. In the worst case, this ratio can be as high as $\log n$.

4.3 Local Search

4.3.1 Simualted Annealing

Simulated Annealing (SA) can be considered as a versin of an “iterative improvement algorithm” which considers only better neighbours and terminates in the first local minima. Unlike this algorithm, simulated annealing use the neighbours of a solution as a way to explore the solutions space and although they prefer better neighbours they also accept worse neighbours in order to avoid getting stuck in local optima. As a result, if the algorithm is run for an infinite amount of time, the global optimum will be found.

Historically, simulated annealing is important and has interesting theoretical properties, however, it achieves good performance at the cost of substantial run-times. We actually has this simple simulated annealing implemented in the first place, but its performance is relativel poor. No convergence is achieved for a relatively large graph at a reasonable amount of time. To improve the performance, we did some literature review and find an Efficient Simulated Annealing (ESA) algorithm for minimum vertex cover.

For the minimum vertex cover problem it is well know that a vertex which has a larger degree that the other vertices will be put into the cover with higher probability because such a vertex can cover more edges. First, the cost function for a given candidate is defined as as follows:

$$Cost = \sum_{i=1}^{n} v_i + \sum_{i=1}^{n}\sum_{j=1}^{n} \overline{v_i \wedge v_j}$$

where the state of the ith vertex can be determined by

$$v_i = \begin{cases} 1 & \text{if the vertex i is in the cover,} \\ 0 & \text{otherwise} \end{cases}$$

Notice that the cost of a given candidate gives the vertex cover of size equal to the cost. This is beacuse we can always construct a vertex cover based on the current cover set by include a vertex from each of the uncovered edges.

Based the previous intuition, we modify the acceptance criteria as follows

$$p = \begin{cases} e^{-\frac{\Delta(1-Deg(v_i))}{T}} & v_i = 1 \quad (1a) \\ e^{-\frac{\Delta(1+Deg(v_i))}{T}} & v_i = 0 \quad (1b) \end{cases}$$

$$Deg(v_i) = \frac{Degree(i)}{EdgeNum}$$

where $Degree(i)$ is the degree of vertex i, its value is equal to the number of edges linking to the ith vertex, and $EdgeNum$ is a constant, equal to total number of edges in a given graph.

ESA starts with an initial solution, C. A neighbour C' to this solution is generated by changing the state a random vertex. If a reduction in cost is found, the current solutino is replaced by the generated neighbour. Otherwise, we use Eq.1, to determine whether a solution is replaced by its neighbour. By using Eq.1, we have

1. If $v_i = 1$, this means v_i is originally excluded from the cover set. From Eq. 1a, p takes a larger value if the degree of the ith vertex is larger, which means C' will be accepted as a new solution with higher probability. That is to say, a vertex with larger degress will be selected into the cover set with higher probability.

2. If $v_i = 0$, this means v_i is orginally included in the cover set. From Eq. 1b, p takes a smaller value if the degree of the ith vertex is larger, which means C' will be accepted as a new solution with lower probability. That is to say, a vertex with a larger degree will be removed from the cover set with lower probability.

4.3.2 *Two Stage Exchange with edge weight forgetting*

Following the papar's content we develop the method of the local search problem with the 2 stage exchange and edge

Algorithm 4: Simulated Annealing

```
Input  : graph $G = (V, E)$, the $cutoff$ time
Output: vertex cover of $G$
1  Construct initial vertex cover $current$;
2  while $elapsed < cutoff$ do
3      T := schedule(t);
4      $next$ := Random-Successor (reverse the state of a random vertex $v_i$);
5      $\Delta$ := cost(next) - cost(current);
6      if $\Delta < 0$ then
7          current := next;
8      else
9          if $v_i = 1$ then
10             current := next with probability Eq. 1a
11         else
12             current := next with probability Eq. 1b
13         end
14     end
15 end
16 return $C$
```

weight with forgetting techniques mentioned in the paper. During the implementation process we find that the weight forgetting technique could be slightly improved as we could add change the weight of the chosen edge when it is once again inserted back into the vertex cover. This could make the recently changed edge once again get lower weight and making the nodes that has been choose less prone to be chosen for the next time. This would slightly improve the algorithm, but there would exist a threshold and a regression ratio as mentioned in the paper that would be the best choice for the choosing of the vertex during the exchange stage. Our next stage would be to carry out experiment to determine the best combination of these parameters. And made comparison with the algorithm that the author original described.

The score for the calculation would be

$$Score(G, u) = cost(G, X) - cost(G, X')$$

where

$$cost(G, X) = \sum_{e \in \text{G and not coverd by X}} w(e)$$

and $X' = X \setminus U$ if $u \in X$, and if not the otherwise union.

The initialization of the vertex cover would pose great influence on the iteration step of the local search algorithm. We currently use the greedy initialization and approximation initialization. The result would be different as the greedy sometime would lead to the local minimal problem and the local search is not easy to get out of local minimal as the perturbation strategy are quite different and would always lead to different conclusion.For the approximation of the initialization, we obeserved a quit great result as the local search strategy would converge to the result that got good quality with respect to the optimal solution.

One improvement we did to the algorithm is the weight forgetting mechanism. From the paper it would decrease the total weight of the graph if the overall mean edge weight reaches some threshold,here we applied more aggressive weight forgetting policy as we set the weight to multiply by some factor to decrease it once it reaches some threshold. The ra-

Algorithm 5: Two Stage Exchange with weight Forgetting

```
    Input   : graph $G = (V, E)$, the $cutoff$ time
    Output: vertex cover of $G$
 1  initialize vertex cover $C$ using approximation or greedy;
 2  result $C^* \leftarrow C$;
 3  uncovered graph $G^* \leftarrow \{\}$;
 4  initialize weight for $E \in G$;
 5  while $elapsed < cutoff$ do
 6      if $G^*$ is empty then
 7          $C^* := C$ ;
 8          remove a vertex with minimum score from $C$;
 9          continue;
10      end
11      $U :=$ choose the neigbor with the largest calculated score;
12      $C = C \setminus U$;
13      subgraph $G^* = G^* \cup N(U)$,where $e \in N(U)$ is is the edge that no longer covered because of removal of U ;
14      choose $V$ of the largest score, where $V \in G^*$ and $V \neq U$;
15      get rid of edges in $G^*$ that got one end of V,adjust their weight;
16      $C = C \cup V$;
17      w(e)=w(e)+1 for $e \in G^*$;
18      if $w(e) > \gamma$ then
19          $w(e) = \lfloor w(e)^* \times \alpha \rfloor$
20      end
21  end
22  return $C^*$
```

tional behind this is that as each time only small number of the edges are in uncovered state, so the focus should be on them rather than the whole and should be treate differently to decrease its impact on the VC once one of its nodes are choosed to be in the VC.

5. EMPIRICAL EVALUATION

5.1 Platform

- CPU: 2.5GHz Intel Core i5
- Memory: 4GB 1600 MHz DDR3
- Language: C++
- Compiler: GNU C++

5.2 Experimental Procedure

For each graph, we run the test for 10 times with different random seed. The results for the Comprehensive Table below are the average of the 10 runs. As for evaluation plots, 40 runs with cutoff set to 3600 seconds are performed for the Efficient Simulated Annealing algorithm, while 100 runs with cutoff set to 60 seconds are carried out for the Two Stage Exchange with Edge Weight Forgetting algorithm. From the the solution trace file, we collect the information needed to generate the plots. The Two stage Exchange with Edge Weight Forgetting algorithm's parameters involves a threshold of weight forgetting, and the decreasing ratio when forgetting these edge weight,in our experiment we set them to be 100 and 0.3 accordingly based on some initial experimental purterbation on power graph.

5.3 Comprehensive Table

The Tables below report the relative error with respect to the optimum solution quality for all algorithms we have implemented. Relative error ($RelErr$) is computed as $(Alg - OPT)/OPT$.

5.4 Evaluation Plot

5.4.1 Simulated Annealing

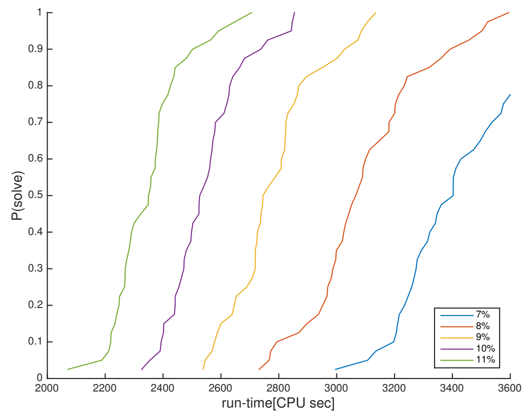

Figure 2: Qualified RTDs for SA on power

From Fig. 2, we can see that for a given solution quality, the probability of finding a desired solution grows the run-time increases. For certain quality, we may not find a solution until significant run-time has passed. If we are trapped in a local minimum, we may not get a near optimal solution at all. On the other hand, for a fixed run-time, the probability of find a desired solution increases as the quality of the solution drops. The best solution we can find also correlates with run-time. For example, the best solution we can find at $T = 3000s$ is one with quality 7%.

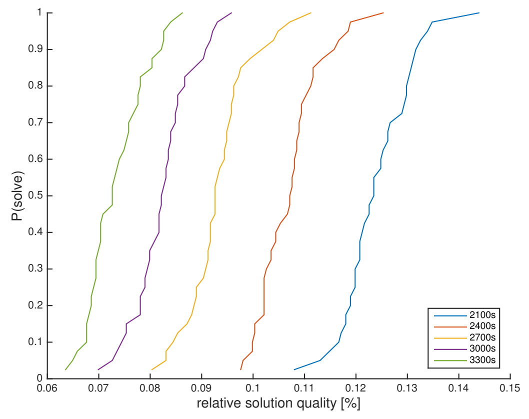

Figure 3: Solution quality distributions for SA on power

	Branch-and-Bound algorithm		
Dataset	Time (s)	VC value	RelErr
jazz.graph	1800	170	0.07
karate.graph	0.049	14	0
football.graph	1800	96	0.02
as-22july06.graph	1800	5684	0.72
hep-th.graph	1800	5295	0.34
star.graph	1800	10774	0.56
star2.graph	1800	6344	0.40
netscience.graph	1800	1085	0.21
email.graph	1800	786	0.32
delaunay n10.graph	1800	827	0.17
power.graph	1800	3188	0.45

Table 1: Performance of Branch-and-Bound algorithm

	Edge Deletion			Maximum Degree Greedy		
Dataset	Time (s)	VC value	RelErr	Time (s)	VC value	RelErr
jazz.graph	2.39e-4	194	0.23	6.87e-3	159	6.33e-3
karate.graph	6.6e-5	22	0.57	1.59e-4	14	0
football.graph	1.55e-4	110	0.17	1.98e-3	96	0.02
as-22july06.graph	1.17e-2	5684	0.72	5.06	3307	1.21e-3
hep-th.graph	5.15e-3	5678	0.45	3.48	3944	4.58e-3
star.graph	5.77e-3	10774	0.56	9.68	7374	0.07
star2.graph	1.16e-2	6344	0.40	6.10	4697	0.03
netscience.graph	8.72e-4	1224	0.36	0.17	899	0
email.graph	7.19e-4	816	0.37	0.10	605	0.02
delaunay n10.graph	5.7e-4	956	0.36	0.11	737	0.05
power.graph	2.53e-3	3736	0.70	1.20	2277	0.03

Table 2: Performance of Approximate algorithms

	Efficient Simulated Annealing			Two Stage Exchange		
Dataset	Time (s)	VC value	RelErr	Time (s)	VC value	RelErr
jazz.graph	3	158	0.01	0.08	162	0.04
karate.graph	0.03	14	0	1e-3	14	0
football.graph	28	94	0	0.06	94	0
as-22july06.graph	3600	5450	0.65	140	3320	5.1e-3
hep-th.graph	3600	4475	0.14	36	4007	0.02
star.graph	3600	9507	0.38	299	7055	0.02
star2.graph	3600	5858	0.29	248	4635	0.02
netscience.graph	384	899	0	2.98	899	0
email.graph	990	595	1.6e-3	1.86	598	6.7e-3
delaunay n10.graph	1466	706	4.3e-3	2.73	713	0.01
power.graph	3600	2328	0.06	36	2205	9.1e-4

Table 3: Performance of Local Search Algorithms

The Fig 3 shows that for a given run-time, the higher the soution quality, the smaller the probability we will get the desired solution. On the other hand, the probability of getting a solution of certain quality increases as the run-time goes up.

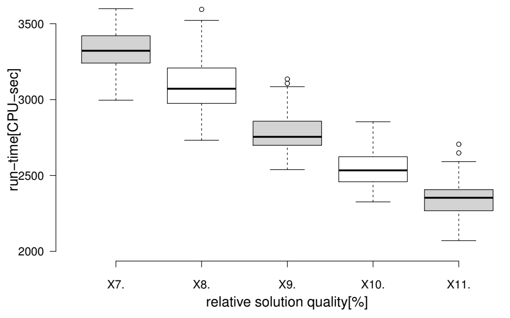

Figure 4: Boxplot for SA on Power

Since the local search algorithm are randomized, there will be some variation in the running times. Boxplots show this variation. For a given quality, the running time needed to achieve that quality varies in each run. From Fig. 4, we can see that there exists one or two outliers in terms of the running time, but the variation is not too much.

Basically, the same conclusion can be drawn from Fig 5 to 7. The longer the run time, the higher the probability you will get the solution of desired quality. On the other hand, the higher the solution quality you want, the longer the run time might be. There is some variation in the run time for a given quality, but the outlier happens only occasionally.

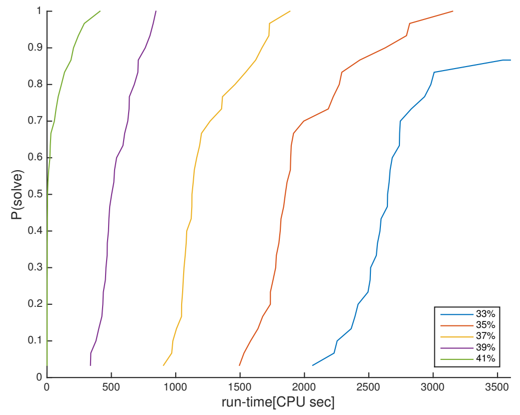

Figure 5: Qualified RTDs for SA on star2

5.4.2 *Two Stage Exchange with Edge Weight Forgetting*

For the QRTD Fig 8 shows the distribution of the runtime of the LS2 algorithm, we could observe that as the algorithm is efficient,when provide larger quality criteria, most of the runs would converge towards it in relatively small amount of time, which is identically shown for the plot with 4% qualtily,and we also observed the plantform shape in the

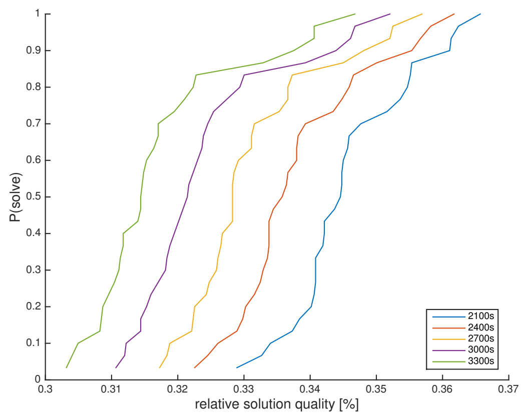

Figure 6: Solution quality distributions for SA on star2

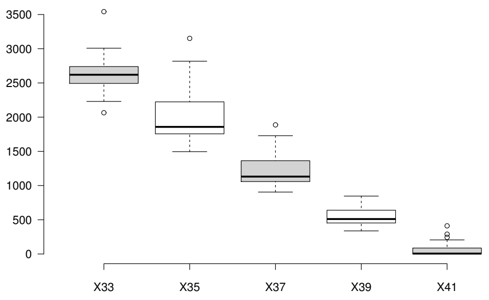

Figure 7: Boxplot for SA on star2

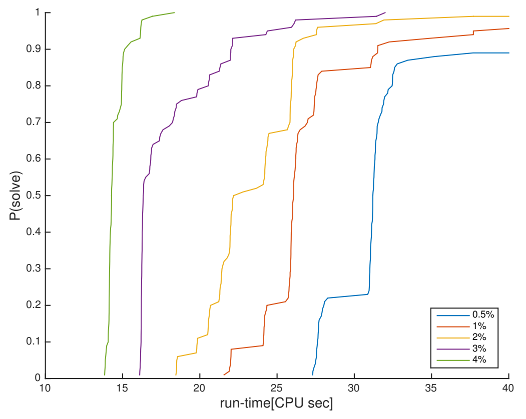

Figure 8: Qualified RTDs for 2 Stage Exchange on power

plotting with lower quality criteria, which means that the schocastic exchange is indeed not relative random,the reason for this is due to some specific node in the exchange state is critical, which would require to be in the sets to make the complete VC,so this lead to some uniform behavior for the 100 times runs, which is shown as the plantform shape in the plotting, as the quality is good and the power graph is relatively medimum size,so changes in the quality in small number would lead to small changes in the selection criteria when plotting, that is one reason for the similar shape for the plotting 0.5%,1%,2%.

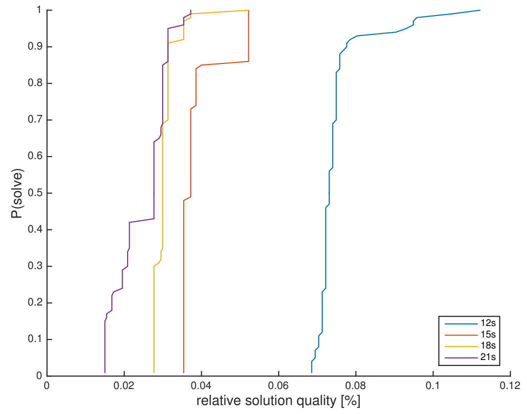

Figure 9: Solution quality distributions for 2 Stage Exchange on power

As the Fig 9 Show the distribution of the quality given a cuttoff time for the power graph by running the second local search algorithm, as the local search algorithm use two stage exchange strategy, it is relatively fast compared with the SA local search, and as the power graph is relatively small, so we could observe a huge increase in the percentage of the solution runs when given a runtime. From the trace file we could see that even with the initialization quality of 70%, this algorithm could converge towards the optimal pretty fast,that is how given some relatively large quality, the plot would be a nearly vertival line,that is why the huge gap between the plot with 12s and other plot,which indicates most of the runs would achieve relatively good quality after 12s of runtime. Another observation we could get is the plantform on the plot,which means the algorithm would fast converge toward the optimal and slightly improves,this could be seen from the 12s and 15s plotting on the graph.

The boxplot for the second local search algorithm is shown in Fig 10, from the graph we could observe that,for most runs to achieve solution with 4% quality,the time varies little from 12s to 14s ,which makes the box small and other statics like median and 3rd quartile and 1st quartile lies in a small range.And for the solution with 0.5% quality,the range for the maximum and minimal is relatively large and we could see the time distributes alone the 100 times runs as the 3th quartile and 1st quartile varies to make the plot looks larger.Also the median is almost in the 3rd quartile,which mean the uneven distribution in the runtimes for this quality. The reason is explained in the QRTD plot's explanation.

The Fig 11 indicates the distribution of the runtime given the quality for the 100 times run of the second local search

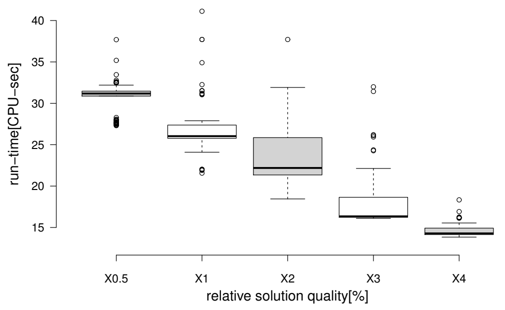

Figure 10: Boxplot for 2 Stage Exchange on power

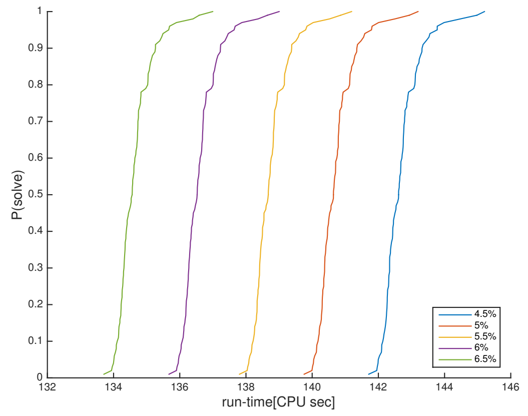

Figure 11: Qualified RTDs for 2 Stage Exchange on star2

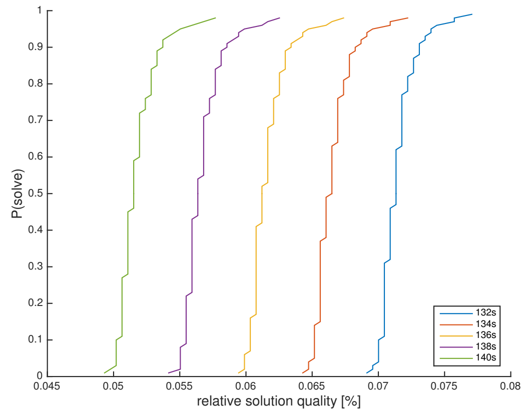

Figure 12: Solution quality distributions for 2 Stage Exchange on star2

algorithm. The similar shape for the plotting indicates that the algorithm decreases steady towards the optimal, this is due to the mechanism of the algorithm that each run

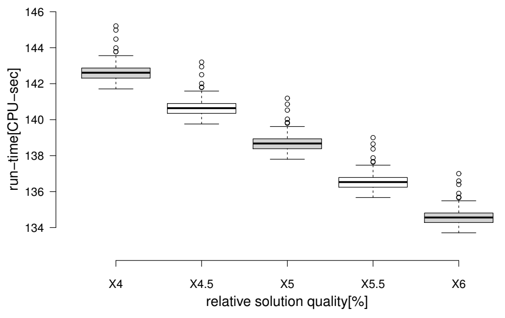

Figure 13: Boxplot for 2 Stage Exchange on star2

it would exchange the node from the current selected sets with the highest score.And only when the solution is near the optimal the convergence would be different due to the different solution.Fig 12 indicates the phenomeno as well as the algorithm is quite steady and the actually perturbation on the choice strategy for the algorithm would not affect its node selection.And another reason for this plotting to appear to be steady compared with the plotting of power, which would lead to large amoutn of variation due to small size of the graph and tiny changes in the plotting criteria.An from these plotting on the large size graph we could conclude that the second local search algorithm is steady and with great efficiency.

6. DISCUSSION

6.1 Branch-and-bound

As we implement 2 methods for branch-and-bound: both recursion and iterations and the results only shows the iterations. But we will discuss why we do not use recursion method. By running the Karate.graph, we could see that the recursion took 5.2ms and iteration took 38.9ms. But when applying large graph the recursion method leads to the stack overflow problem eventhough the recursion runs faster then iteration. Overall, the branch-and-bound method takes a very long time to find the optimal solution, because the time complexity is still $O(V!)$ even if we use upper bound and lower bound to prune the solutions. From table 1, we could see that the performance of Branch-and-Bound is at least better than approxmation method, because for the initial condition, we use the solution of approximation mathod as upper bound. Besides, Brach-and-Bound method took a long time to run except for the karate graph which the relative error is 0.

6.2 Approximation Algorithms

Although the Maximum Degree Greedy Algorithm has an approximation ratio of $\log n$, which is much higher than the Edge Deletion's ratio of 2, in practice it usually achieves a better solution. In particular, in all the given datasets, MDG significantly outperforms ED. The reason is that ED needs to take care of those very rare cases, which one usually would not see in real-world datasets. MDG, on the other hand, by giving up those rare cases, it can achieve a better solution for common cases.

6.3 Local Search with Efficient Simualted Annealing

The key idea in Efficient Simulated Annealing is that the a vertex which has larger degree that the other vertices will be put into the cover with higher probability. Thus, instead of using a single acccptance function, we use two different acceptance funcitn for different cases in the hope that retaining the vertices with larger degree in the current cover set and select vertices with larger degree into the current cover set. During experimenation, we do find it out-perform the standard simulated annealing, especially for the larger graphs. However, it still takes substantial running time to achieves convergence. For example, we can not get near optimal solution for large graphs like as-22july06.graph, star2.graph within an hour. The good news is that we obeserve that it do get out local optimum with more likely, resulting in a better solution than other algorithms in the local search family.

6.4 Local Search with two stage exchange with edge weight forgetting

The rationale behind this algorithm was the smoothing effect of the weight of the edge and its influence would result in a faster algorithm with better convergence quality. Also the two stage exchange mechanism would relieve the huge computation in the determination in the relationships between the two nodes, thus seperate operation in the choosing and adding vertices would result in better performance, and the way the criteria of the vertices to be choosen is also related to the edge weight,which the two related operation would together contribute towards the convergence of the result. Also, based on the algorithm purposed by the author, we also changed the edge weighting forgetting mechanisms as decreasing the edge's weight when adding its attached nodes to the current selection, which would make the dcore for the nodes to decrease in the next round,making it less likely to be repeatly choosen for upcomming rounds, this would lead to slightly improvement as the way the original work proposed, which would take the overall average edge's weight into consideration. The mechanism behind this is similar and we observerd similar result, more experiment could be conducted to compare out algorithm with the original one in the aspect of the decreasing rate of edge weight, which would pose great influence in node selection stage. As in our project the focus would be on the different algorithms, so detailed comparison would be less concerned and our implementation just choose the threshold of the edge forgetting start value to be 100 and decreasing ratio to be 0.3, for which we could get best result from the experiments on the power graph.As this setting would actually differs from different size of the graphs, more comprehensive invistigation could be applied as another interesting topic.

Another intersting observation would be the different initialization and their impact on the convergence of the result.We adopted greedy and approximation approach to get the initial solution, and saw different behavior on the convergence of the computation. Greedy initialization would be more likely to lead to a local minimal while for approximation we saw the good behavior of convergence. Quantilized comparison with different initialization method would also be an interesting topics for the future work.

7. CONCLUSION

We presented four different algorithms for the classical Vertice-cover problem,by adopting the branch and bound, approximation, and two local search strategies for them, as we could see from the result of these algorithms,approximation approach would get the result no more than two times as the optimal one, and with a guaranteeing a worst case approximation ratios of 2. Branch and Bound algorithm would require lots of computation time and space, but would return accurate given enough computation capability. For the local search algorithm, the result would be useful as it could yield better result compared with approximation and run in less time compared with the Branch and Bound. Through the experiment of the two local search algorithm we could say that Simulated Annealing and Two stage exchange with Edge Weight Forggeting are stable with efficiency.

8. REFERENCES

[1] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*, MIT Press, 2010.

[2] Cai et al., 2013. Shaowei Cai, Kaile Su, Chuan Luo, and Abdul Sattar. NuMVC: An efficient local search algorithm for minimum vertex cover. *J. Artif. Intell. Res. (JAIR)*, 46:687-716, 2013.

[3] Cai, S. 2015. Balance between complexity and quality: Local search for minimum vertex cover in massive graphs. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence* 25-31, 2015, pp. 747-753.

[4] X.S. Xu, J. Ma. An efficient simulated annealing algorithm for the minimum vertex cover problem. *Neurocomputing*, 69 (2006), pp. 913-916.

[5] R. M. Karp. Reducibility among combinatorial problems.In *Complexity of Computer Computations*, pp. 85-103, 1972.

[6] D. Johnson. Approximation algorithms for combinatorial problems. *J. Computer and System Sciences*, 9:256-278, 1974.

[7] D. Hochbaum, editor. *Approximation Algorithms for NP-hard Problems.* PWS Publishing, Boston, 1996.

[8] V. Paschos. A survey of approximately optimal solutions to some covering and packing problems. *Computing Surveys*, 171-209, 1997.

[9] V. Vazirani. *Approximation Algorithms.* Springer, 2004.

[10] F. Delbot and C. Laforest. Analytical and experimental comparison of six algorithms for the vertex cover problem. *Journal of Experimental Algorithmics (JEA)*, 15:1-4, 2010.

[11] F. Gomes, C. Meneses, P. Pardalos, and G. Viana. Experimental analysis of approximation algorithms for the vertex cover and set covering problems. *Computers and Operations Research*, 33:3520-3534, 2006.

[12] T. Grossman and A. Wool. Computational experience with approximation algorithms for the set covering problem. European J. Operational Research, 101, 1997.

[13] S. Richter, M. Helert, and C. Gretton. A stochastic local search approach to vertex cover. *In Proc. 30th German Conf. on Artificial Intelligence*, 2007.

[14] S. Khot and O. REGEV. Vertex cover might be hard to approximate to within $2 - \epsilon$. *J. Comput. Syst. Sci.* 74, 3, 335-349.