Minimum Vertex Cover

Hongyu Chen
Georgia Institute of Technology
Atlanta, United States
hchen687@gatech.edu

Rui Jia
Georgia Institute of Technology
Atlanta, United States
rjia40@gatech.edu

Xiaotong Mu
Georgia Institute of Technology
Atlanta, United States
xmu33@gatech.edu

Xiangyu Liu
Georgia Institute of Technology
Atlanta, United States
xliu737@gatech.edu

ABSTRACT

The Minimum Vertex cover (MVC) problem is a well known NP-complete problem with numerous applications in computational biology, operations research,the routing and management of resources. In this project, we will treat the problem with Branch and Bound Algorithm, Approximation Algorithm and two families of Local Search Algorithms. Then we will evaluate their theoretical and experimental complexities on real world datasets.

KEYWORDS

Algorithms, NP-Complete,Minimum Vertex Cover, Branch and Bound, Approximation, Local Search, Simulated Annealing

1 INTRODUCTION

In the mathematical discipline of graph theory, a vertex cover (sometimes node cover) of a graph is a set of vertices that includes at least one endpoint of every edge of the graph. The problem of finding a minimum vertex cover is a classical optimization problem in computer science and is a typical example of an NP-hard optimization problem.
Bound Algorithm, Approximation Algorithm and Local Search Algorithms are used to solve minimum vertex cover problem.

2 PROBLEM DEFINITION

Formally, a vertex cover V' of an undirected graph G=(V,E) is a subset of V such that uv ⊂ E => u ⊂ V' and v ⊂ V', that is to say it is a set of vertices V' where every edge has at least one endpoint in the vertex cover V'. Such a set is said to cover the edges of G.
A minimum vertex cover is a vertex cover of smallest possible size. The vertex cover numberτ is the size of a minimum vertex cover, i.e.τ =|V'|. The following digram shows an example of minimal vertex cover with the solution represented by {a,c,f,g}

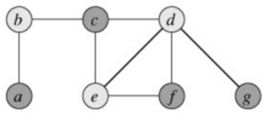

Figure 1: Example of Minimum Vertex Cover

The minimum vertex cover problem is the optimization problem of finding a smallest vertex cover in a given graph.We input a Graph G, and get the smallest number k such that G has a vertex cover of size k.

3 RELATED WORK

Several heuristics and approximations algorithms have been proposed in solving the Minimum Vertex Cover problem. Chleb [7] proposed a branch-and-bound algorithm in finding near optimal solutions which explores the configuration space by deciding about the presence or not of one node by the cover in each step of the recursion and recursively solving the problem of the remaining nodes. Covered node and all adjacent edges are removed, while an ignored node remains, but cannot be selected in deeper levels of the recursion. Subsets of nodes that provide valid vertex covers are identified and the smallest of them is the minimum vertex cover. Branch-and-bound is a general algorithm for finding optimal solutions of various optimization problems, especially in discrete and combinatory optimization. It is easy to see that in the worst case, the complexity of branch-and-bound is upper bounded by the total number of nodes in the recursion tree, which is proportional to 2^n.

When we are working on an optimization problem in which each potential solution has a positive cost, we wish to find a near optimal solution. Traditionally, approximation ratio is been used to measure the performance. We say that an algorithm for a problem has an approximation ratio of ρ(n) if, for any input of size n, the cost c of the solution produced by the algorithm is within a factor of ρ(n) of the cost c* of an optimal solution:

$$\text{Max}\left(\text{c}/\text{c}^*, \text{c}^*/\text{c}\right) \leq \rho(\text{n})$$

For approximation algorithm, since this is a minimization problem in our study, we are interested in smallest c/c^*, and it's been proved that $c/c^* \leq 2 = p(n)$, in other words, the approximation vertex cover algorithm returns a vertex cover that is at most twice the size of an optimal cover
With traditional heuristics for vertex cover[9], the most obvious is the greedy algorithm, in which the vertex of maximum remaining degree is repeatedly removed from the graph and added to the cover until all edges are covered. While intuitive, the algorithm actually performs poorly on many classes of graphs and has no fixed performance bounds. An alternative algorithm is focused on finding a maximal matching in the graph: while edges remain, choose an arbitrary edge, add both endpoints to the cover, and remove both vertices from the graph. The algorithm has a fixed performance bound of 2, because each selected edge must be covered by at least

one of its endpoints.
Recently, there is increasing interest in solving Minimum vertex cover on massive graphs. Since the introduction of FastVC.The main principle of FastVC is to use low- complexity approximate heuristics rather than those accurate heuristics with high complexity.FastVC solves the MinVC problem by iteratively solving its decision version. For the exchanging step in local search, FastVC adopts the two-stage exchange framework, as it has lower complexity than the alternative paradigm based on vertex pair exchange.

4 ALGORITHMS

4.1 Branch and Bound

4.1.1 Description. Brand and Bound is an algorithm for discrete and combinatorial optimization problems, as well as mathematical optimization. A branch-and-bound algorithm consists of a systematic enumeration of candidate solutions by means of state space search: the set of candidate solutions is thought of as forming a rooted tree with the full set at the root. The algorithm explores branches of this tree, which represent subsets of the solution set. Before enumerating the candidate solutions of a branch, the branch is checked against upper and lower estimated bounds on the optimal solution and is discarded if it cannot produce a better solution than the best one found so far by the algorithm. Data Structure: HashSet, ArrayList.

Algorithm 1 Branch and Bound

```
$G \leftarrow (V, E)$
$output \leftarrow vertexCover$
$C \leftarrow emptySet$
while !E.isEmpty() do
    Select u,v from E
    $C \leftarrow C \cup \{u, v\}$
    $v \leftarrow v - \{u, v\}$
    $E \leftarrow E$ all edges covered by {u,v}
end while
return $C$
```

4.1.2 Pseudo Code.

4.1.3 Algorithm Performance. We used the backtracking to find the best result, which result to an exponential run time complexity. The Big-O time complexity for this algorithm is 2^n. n is decided by the number of vertex. To improve the algorithm, we applied memorialized backtracking to store the explored vertex. Therefore, every branch has two child nodes and the maximum depth is the number of vertex.
The space comlexity is O(|V|+|E|).

4.2 Approximation Algorithm

4.2.1 Description. One can find a factor-2 approximation by repeatedly taking both endpoints of an edge into the vertex cover, then removing them from the graph. Put otherwise, we find a maximal matching M with a greedy algorithm and construct a vertex cover C that consists of all endpoints of the edges in M.
The set C constructed this way is a vertex cover: suppose that an edge e is not covered by C; then M e is a matching and e M, which is a contradiction with the assumption that M is maximal. Furthermore, if e = u, v M, then any vertex cover – including an optimal vertex cover – must contain u or v (or both); otherwise the edge e is not covered. That is, an optimal cover contains at least one endpoint of each edge in M; in total, the set C is at most 2 times as large as the optimal vertex cover.

(1) Initialize the result as empty list
(2) Consider a set of all edges in given graph. Let the set be E
(3) Do following while E is not empty : a) Pick an arbitrary edge (u, v) from set E and add 'u' and 'v' to result. b) Remove all edges from E which are either incident on u or v.
(4) return result

Algorithm 2 Branch and Bound

```
$G \leftarrow (V, E)$
$output \leftarrow edgeNumber$
$C \leftarrow emptySet$
$E' \leftarrow$E (uncovered edges)
while !E.isEmpty() do
    Select u,v from E
    $C \leftarrow C \cup \{u, v\}$
    $v \leftarrow v - \{u, v\}$
end while
return $C$
```

4.2.2 Pseudo Code.

4.2.3 Algorithm Performance. The Big-O run time complexity of this algorithm is O(|V|+|E|). In the worst case, we need to include all vertex in cover set and delete all edges. In this case, we need to walk through all vertex and edges.
The space complexity of this algorithm is O(|V|). We used a hashset to record all covered vertex and an array to record the remaining edges numbers for each vertex. Thus, the overrall space complexity is O(|V|).

4.3 Local Search

4.3.1 Description. :
Local Search Algorithm is one kind of heuristic algorithms used to solve NP-hard problems. This kind of algorithm can find a solution quickly. However, it will not guarantee the the solution is optimal. In real word, solution given by the local search will be closed to the optimal one, which makes it has important practical meanings.
The main idea of Local Search Algorithm Framework is listed as Follow from the project descriptionPlease refer to Algorithm1 [4]

In this paper, we discuss about two kind of Local Search algorithms. The first one is called ***FastVC*** , which was proposed by[2]. The second one uses the Simulated Annealing Algorithm.

4.3.2 Local Search 1. :
4.3.2.1 Description
Here we adopt the ***FastVC*** proposed by [2]. In this algorithm, they define two important variables: ***loss(v)*** and ***gain(v)***.[5]
loss(v) means that for a vertex v, how many covered edges will become uncovered after we remove v from our vertex cover set.

Algorithm 3 Local Search Framework

```
$C \leftarrow InitialVertexCover$
while elapsed time < cutoff do
    if $C$ is a vertex cover then
        $C^* \leftarrow C$
        Remove one or more than one vertices from C
    end if
    $u \leftarrow$ Select exiting vertex from C
    $C \leftarrow C \backslash \{u\}$
    $v \leftarrow C$ select entering vertex from V\C
    $C \leftarrow C \cup \{u\}$
end while
return $C^*$
```

gain(v) means that for a vertex v, how many uncovered edges will become covered when we add v to our vertex cover set.

The ***FastVC*** uses its own ***ConstructVC*** procedure to get the initial solution and attain the initial loss(v) for each vertex.Then it initialize the gain(v) as 0 for all the remaining vertex. As long as we have a valid cover set, we keep removing vertex with lowest loss value in C until it's not a vertex cover set.Then we use the Best from Multiple Selections (BMS) method to build their ***ChooseRmVertex*** to reach the neighbors. We repeat this process until time off and return the cover set.

Algorithm 4 FastVC(V,E,cutoff,seed)

```
$C := ConstructVC(V, E)$
$gain(V_i) = 0$ for each $v \notin C$
while elapsed time < cutoff do
    if C cover all edges then
        $C^* := C$
        select a vertex from C with minimum loss then remove it
    end if
    $u := ChooseRmVertex(V, K, loss, seed)$
    $C := C \backslash u$
    e := an uncovered edge selected with seed
    v := endpoint with higher gain in e
    $C := C \backslash v$
end while
return $C^*$
```

ConstructVC: We first initialize our set as empty and set all vertexes' loss value as 0. Then we iterate through all edges in the graph. For each uncovered edge we will add the endpoint which has higher degree into our cover set and update this endpoint's loss value in our set. After iteration, we remove all the vertexes with 0 loss value and update their neighbors' loss value.

ChooseRmVertex: Local search algorithm widely adopt the "best-picking" heuristic since this heuristic can guide our searching to promising area. Recent sample may include max-gain pair selection heuristic (Richter et al., 2007) [8] and in (Cai et al., 2013)[3], it uses the minimum loss removing heuristic in NuMVC. However, this "best-picking" might now work well on a massive data sets since finding the best element is time consuming. [1] And our Best from

Algorithm 5 ConstructVC(V,E)

```
$C := \emptyset$
for each edge ∈ E do
    if edge is uncovered then
        v = $argmax$ degree($v_i$)$v_i \in V$
        $C \leftarrow C \cup$ v
    end if
end for
for each v ∈ C do
    $loss(v) := 0$
end for
for each edge ∈ E do
    if only one endpoint of edge is in C then
        loss(v)++ for the endpoint in C
    end if
end for
for each v ∈ C do
    if loss(v) = 0 then
        $C := C \backslash v$
        Update loss of vertexes in C
    end if
end for
return $C$
```

Multiple Selections (BMS) works by selecting 50 elements randomly and choose the edge with the lowest loss value and return it back to the main algorithm.

Algorithm 6 ChooseRmVertex

```
Require: vertex set V, integer k = 50, loss, seed
    $best$ := random selected element with seed
    for iterantion until k do
        r ← random selected vertex with seed from V
        if loss(r) < loss(best) then
            $best \leftarrow r$
        end if
    end for
    return $best$
```

4.3.2.2 Data Structure
we use HashMap, ArrayList, HashSet from Java.
4.3.2.3 Time and Space Complexity
The time complexity for ConstructVC(V,E) is O(E) since there's no inside loop in each loop and it loop through edges twice. The Space complexity is O(V) since we can use a HashMap to store the cover set and the loss corresponding to each vertex.
The time complexity for ChooseRmVertex(V,k,seed) is O(1) since k is a constant and it's always smaller than the |V|. And the space complexity is also O(1) since we just need to store the vertex we pick.
The time complexity for FastVC is mainly locate in the While loop: the checking process takes O(E). After, it takes O(E) time and space to choosing an uncovered edge randomly. Therefore, the total time complexity is O(V+E) and the space complexity is O(V+E) for fastVC.

4.3.3 *Local Search 2.* Simulated Annealing Description :

Simulated Annealing is an effective and general form of optimization. It is useful in finding global optima in the presence of large numbers of local optima, and it is a fairly robust stochastic optimization algorithm based on local search. The algorithm derives inspiration from the physical process of annealing in metallurgy, which follows a similar analogy to solve optimization problems, by identifying the negative of the quality of the solutions. Unlike the classical local search algorithms that always accept a new better neighbours, Simulated Annealing allows to accept worse solution with a lower probability.

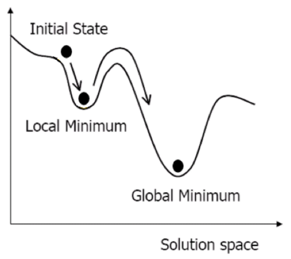

Figure 2: Shows the simulated annealing

Simulated annealing begins with an arbitrary solution and then repeatedly tries to make improvements to it locally. Our decision of accepting the new solution is based on the difference $Q_{\text{old}} - Q_{\text{new}}$ between the quality of the old and new solutions and on T,which is gradually decreasing throughout the process.

As we mentioned previously,for SA the probability of accepting a worse solution is low, that is because the probability of accepting a worse solution from the local step decreases exponentially fast with the ratio of the decrease in solution quality and the T :

$P_{\text{accept}} = \min\{1, \exp((Q_{\text{old}} - Q_{\text{new}})/T)\}$

In the algorithm we used Maximum Degree Greedy (MDG) [6], which is an adaptation of the classical greedy algorithm for the set cover problem. Given a graph G, the algorithm outputs a vertex cover C of G. Initially, C is the empty set. Choose a vertex $v \in V(G)$ of maximum degree, add v to C, and delete v and all edges incident to v from G. Repeat until there is no edge left in G. It's worst-case approximation ratio is $H(\Delta)$, with $H(n) = 1 + \frac{1}{2} + \cdots + \frac{1}{n}$the harmonic series ($H(n) \approx \ln n + 0.57$ when n is large) and Δ the maximum degree of the graph.

To build the Simulated Annealing, we adopt some variables from [10] so as to get to the neighbor solution better. According to the lecture, we set the cooling rate as 0.95 here. This is not the ideal one since it will cool so fast that the algorithm stop accepting new neighbors in a short time.

Algorithm 7 Maximum Degree Greedy (MDG)

```
Require: Input a graph $G = (V, E)$,
  result a vertex cover of $G$
  $C \leftarrow \emptyset$
  while $E \neq \emptyset$ do
      select a vertex $u$ of maximum degree;
      $V \leftarrow V - \{u\}$
      $C \leftarrow C \cup \{u\}$
  end while
  return $C$
```

Algorithm 8 Simulated Annealing

```
Require: Input: graph $G = (V, E)$, cutoff time , seed
  $C \leftarrow MDG(G)$
  while elapsed time < cutoff & $T > 0$ do
    $v \leftarrow$ randomly flip one vertex with the given seed $v_i$
    if $v \in C$ then
      if $C\backslash\{v\}$ still a vertex cover then
        $C' = C\backslash\{v\}$
      end if
    else
      $C' = C \cup \{v\}$
      $\Delta \leftarrow$ F-function $(C')$ − F-function$(C)$
    end if
    if $\Delta < 0$ then
      $C \leftarrow C'$
    else
      $C \leftarrow C'$ with $p_i(vi)$
    end if
    $T = T\times$ cooling rate
  end while
  return $C$
```

$$pi(vi) = \begin{cases} e^{-\Delta(1-deg(v))/T}, v \in C \\ e^{-\Delta(1+deg(v))/T}, v \notin C \end{cases}$$

$$Deg(v) = \frac{degree(v)}{|E|}$$

$$Ffunction = coverset.size()$$

4.3.3.2 Data Structure

we use HashMap, ArrayList, HashSet, Array from Java.

4.3.3.3 Time and Space Complexity

First we take a look into the MDG algorithm, it will loop through the edges once, which makes its runtime as O(E) and the space complexity is O(V). Then the Simulated Annealing algorithm will check if removing this random selected edge will break the cover set by looping through its neighbors. Therefore, the total space complexity is O(V) for the cover set and the time complexity is O(E).

Table 1: Comprehensive Table of Branch and Bound

Graph	Run Time(s)	Result	Rel Error
jazz	0.48	179	13.29%
karate	0,11	14	0.00 %
football	3.07	97	3.19 %
as-22july06	null	null	null
hep-th	0.02	4534	15.48 %
star	null	null	null
star2	null	null	null
netscience	1.59	899	0 %
email	1.16	756	27.27 %
delaunay n10	7.92	933	32.71 %
power	18.11	3246	47.389%

Table 2: Comprehensive Table of Approximation Algorithm

Graph	Run Time(s)	Result	Rel Error
jazz	3.419E-4	164	3.7%
karate	3.6E-5	14	0.00 %
football	6.48E-5	98	4.25 %
as-22july06	0.0320605	5172	56.58%
hep-th	0.0150952	5215	32.82 %
star	0.038331399	8356	21.06 %
star2	0.0251695	6569	44.62 %
netscience	0.0018799	1083	20.46%
email	0.0016427	732	23.23%
delaunay n10	6.74E-4	851	21.05$
power	0.004694199	3282	48.97%

5 EMPIRICAL EVALUATION

5.1 Platform

This experiment is processed by Intel Core i7 processor, with 32GB RAM. And the execution is done by java JDK11 in Windows10. The IDE for this project is IntelliJ IDEA 2019.

5.2 Experiment Results

5.2.1 Branch and Bound. We run the Branch and Bound several times with different cutoff time. Use the cutOff time 200 seconds, we find the result of Branch and Bound algorithm. The result can be found in table 1. Based on the table, we can find than the Bnb algorithm works well on small size graphs. However, with the size of graph increasing, the running time will increase exponentially.

5.2.2 Approximation Algorithm. We run the Approximation several times with different cutoff time. Use the cutOff time 200 seconds, we find the result of Branch and Bound algorithm. The result can be found in table 2. Based on the table, we can find than the approx algorithm works faster than branch and bound algorithm. However, sometimes, it can get really bad accuracy.

5.2.3 Local Search. The devices used for running Local Search are listed as follow:

To get the results, we run each local search method with 10 different seeds for each graph instances with the cutoff as 600 seconds.

Table 3: Description for Platforms

Devices System	Processor	Ram	IDE
1 macOS Catalina	2.7 GHz Intel Core i5	8GB	IntelliJ IDEA
2 Windows 10	2.7 GHz Intel Core i7	8GB	IntelliJ IDEA

Table 4: Comprehensive Table of FastVC

Graph	Run Time(s)	Result	Rel Error
jazz	0.58	158	0.00%
karate	0.00	14	0.00%
football	35.45	94	0.00%
as-22july06	381.79	3312	0.26%
hep-th	199.98	3933	0.19%
star	510.73	6961	0.85%
star2	556.09	4671	2.80%
netscience	0.00	899	0.00%
email	51.74	597	0.53%
delaunay n10	441.99	714	1.60%
power	335.07	2239	1.60%

Table 5: Comprehensive Table of Simulated Annealing

Graph	Run Time(s)	Result	Rel Error
jazz	0.02	163	2.97%
karate	0.00	16	13.33%
football	0.10	97	3.40%
as-22july06	41.34	3326	0.69%
hep-th	8.96	3945	0.48%
star	51.39	7264	5.24%
star2	19.15	4841	6.59%
netscience	0.27	901	0.19%
email	0.36	615	3.58%
delaunay n10	0.30	745	5.99%
power	2.51	2280	3.50%

https://www.overleaf.com/project/5fc97f9b682ad25424c4ac8a We can see from the result that FastVC have a very low relative error, which indicates a great optimization. However, it takes longer time than the Simulated Annealing to get the result.

Although simulated annealing will yield a higher relative error, it use very short time to get to an acceptable results.

5.3 Evaluation Plots

5.3.1 Qualified Runtime. From the Qualified Runtime plot(Fig 3 to Fig 6), the optimal result improve with run time in both algorithms. The discrepancy of the random seed become obvious when the relative error is low. This is because that when the required quality is high, not all test run can reach the barriers. However, when the required quality is low, they can reach it fast. And for simulated annealing, some of the test run will fail to reach the relative error when it's very small, which is consistent with our conclusion before: it take less run time but the result cannot be guaranteed. However, we can see that all the relative errors are below 0.1, which indicates that such solutions are acceptable. And since the cooling process is exponential, it will stop accepting the new solution soon.

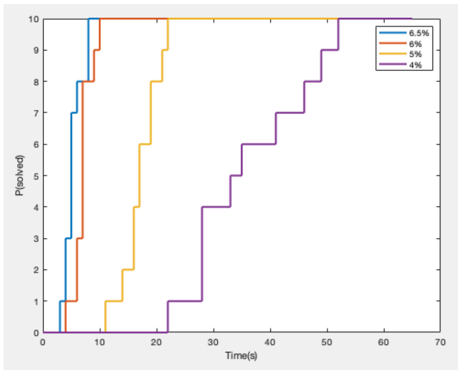

Figure 3: QRTD for FastVC for star2.graph

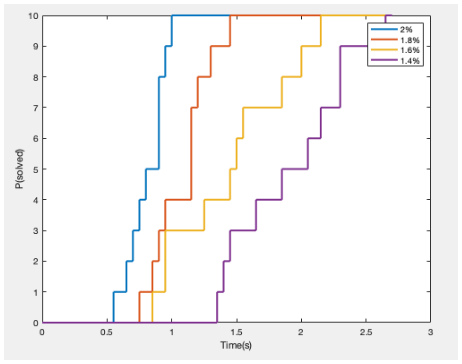

Figure 4: QRTD for FastVC for power.graph

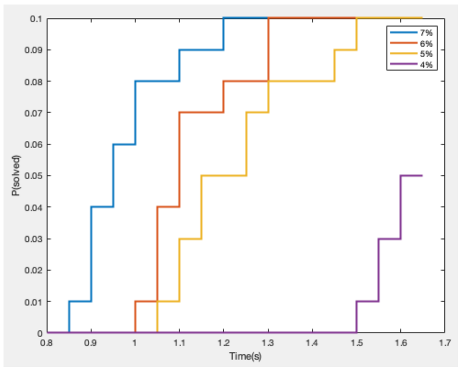

Figure 5: QRTD for Simulated Annealing for power.graph

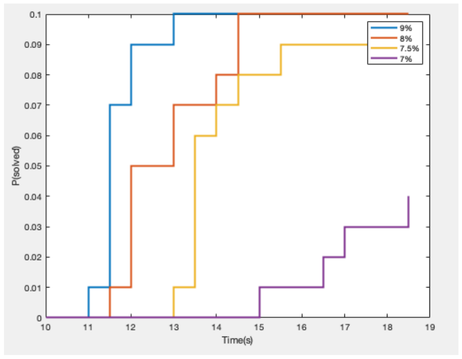

Figure 6: QRTD for Simulated Annealing for star2.graph

5.3.2 Solution Quality Distributions. Our the SQDs results(Fig 7 to Fig 10) indicate that our former conclusion is valid. As the runtime increases, the quality of our solutions will also improves. Also the probability of getting a solution will also increase when the required quality decrease.

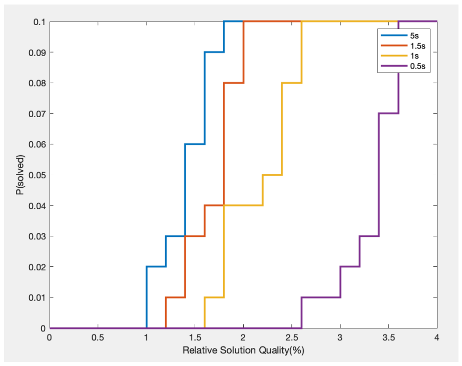

Figure 7: SQDs for FastVC for power.graph

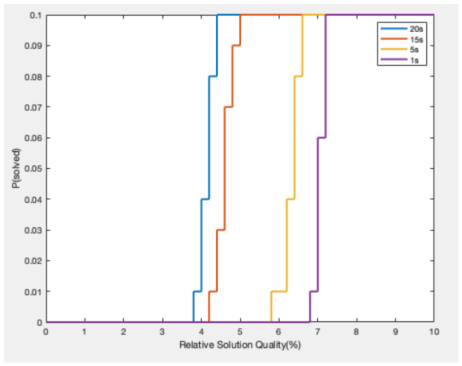

Figure 8: SQDs for FastVC for star2.graph

5.3.3 Boxplot. From plots Fig 11 to Fig 14, we can see that in general, FastVC will have a larger mean and IQR. And with a larger graph, both algorithms require more run time to reach the solution.

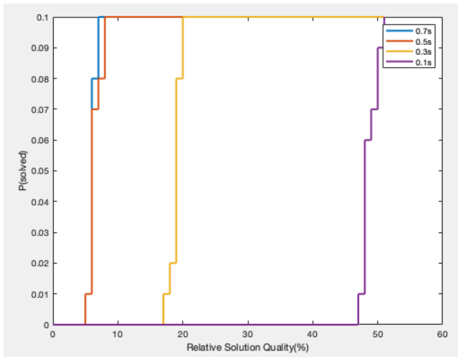

Figure 9: SQDs for Simulated Annealing for power.graph

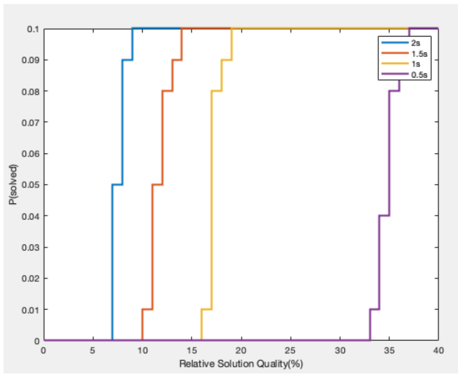

Figure 10: SQDs for Simulated Annealing for star2.graph

Also, the run time for a better quality will also be larger. All these behaviors are consistent with our analysis before.
For FastVC, it's more sensitive to different random seed since there are two random process in its procedure. As for the simulated annealing, since we set the cooling rate as 0.95, it will cool down very soon, making it not to accept new neighbors.

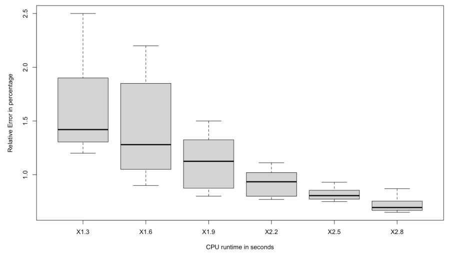

Figure 11: BoxPlot for FastVC for power.graph

6 DISCUSSION

Based on the above experiment results, we can find that the Branch and Bound algorithm performs well when solving the small graph, the error rate can be near 0.00%. However, when the graph becomes larger, the lower bound plays a significant role in pruning

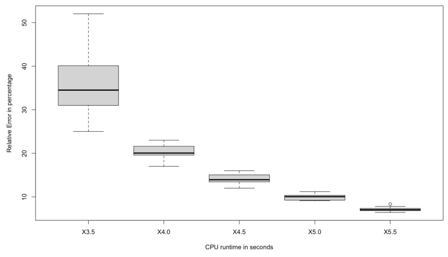

Figure 12: BoxPlot for FastVC for star2.graph

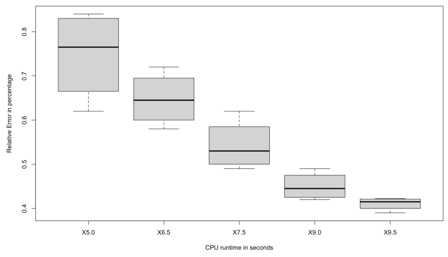

Figure 13: BoxPlot for Simulated Annealing for power.graph

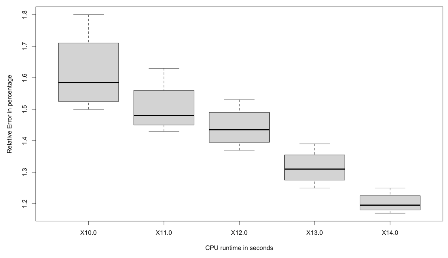

Figure 14: BoxPlot for Simulated Annealing for star2.graph

the branch. The time complexity for the algorithm is exponentially increasing with number of vertex. We used two-approximation to find the lower bound and it is not a tight lower bound so that it causes increasing error rate growing.
For heuristic with approximation, we used the edge deletion algorithm, which showed majority of cases are 20% - 50%. The approximation algorithm is not as sensitive as Branch and Bound algorithm. However it trades the approximating quality for a faster processing time as we can see from the running time in result table among others. It has worse accuracy. In addition, we can observe that the edge deletion algorithm works and it has a better performance on the relatively small graph.
The fastVS and simulation local search algorithms we implemented are mach more efficient and stable. The local search algorithm can get acceptable relative errors. The averaged relative errors of the solutions with FastVC are mostly less than 5%. Actually this is result very closed to the optimal solution. On the other hand, the Simulated Annealing methods returns relative errors higher than

the fastVC algorithm. But the largest error is only near 13% which is also acceptable. However, since the cooling process is exponential, it will stop accepting the new solution soon. Therefore, more work need to be done about finding a suitable cooling rate.

7 CONCLUSION

The minimum vertex cover problem can be solved by Branch and Bound, Approximation Algorithm, and local search algorithm. The branch and bound algorithm has an algorithm with running time exponential in the input size, but which might do well on the small inputs.Approximation algorithm can quickly find a solution that is provably not very bad. By local search algorithm, we can quickly find a solution for which we cannot give any quality guarantee, but which might often be good in practice on real problem instances. The FastVC algorithmn sacrifice the run time in an acceptable range and get a very high quality solution. Compared to other algorithm, the major strength of it is that it's ability to treat massive data sets. And the simulated annealing algorithm get the solution in a short run time. And more work need to be done about finding a suitable cooling rate

REFERENCES

[1] D. Andrade, M. C. Resende, and Renato F. Werneck. 2012. Fast local search for the maximum independent set problem. *Journal of Heuristics* 18 (2012), 525–547.

[2] Shaowei Cai. 2015. Balance between Complexity and Quality: Local Search for Minimum Vertex Cover in Massive Graphs. (2015), 747–753.

[3] Shaowei Cai and Kaile Su. 2013. Local search for Boolean Satisfiability with configuration checking and subscore. *Artificial Intelligence* 204 (2013), 75 – 98. https://doi.org/10.1016/j.artint.2013.09.001

[4] Shaowei Cai, Kaile Su, and Abdul Sattar. 2011. Local Search with Edge Weighting and Configuration Checking Heuristics for Minimum Vertex Cover. *Artif. Intell.* 175, 9–10 (2011). https://doi.org/10.1016/j.artint.2011.03.003

[5] Shaowei Cai, K. Su, and A. Sattar. 2012. Two New Local Search Strategies for Minimum Vertex Cover. (2012).

[6] François Delbot and Christian Laforest. 2010. Analytical and Experimental Comparison of Six Algorithms for the Vertex Cover Problem. 15, Article 1.4 (Nov. 2010), 27 pages. https://doi.org/10.1145/1671970.1865971

[7] Chleb M. and Chleb J. 2004. *Crown reductions for the Minimum Weighted Vertex Cover problem.*

[8] Silvia Richter, Malte Helmert, and Charles Gretton. 2007. A Stochastic Local Search Approach to Vertex Cover. (09 2007). https://doi.org/10.1007/978-3-540-74565-5_31

[9] Satoshi Shimizu, Kazuaki Yamaguchi, Toshiki Saitoh, and Sumio Masuda. 2016. A fast heuristic for the minimum weight vertex cover problem. *2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)* (2016). https://doi.org/10.1109/icis.2016.7550782

[10] Ma J. Xu, X. 2006. An efficient simulated annealing algorithm for the minimum vertex cover problem. *Neurocomputing* 69 (2006), 913–916. https://doi.org/10.1016/j.neucom.2005.12.016