CSE6140 Project Final Report

Siying Cen*
Georgia Institute of Technology
Atlanta, USA
scen9@gatech.edu

Chujie Chen
Georgia Institute of Technology
Atlanta, USA
cj.chen@gatech.edu

Shuran Wen
Georgia Institute of Technology
Atlanta, USA
swen43@gatech.edu

Ren Liu
Georgia Institute of Technology
Atlanta, USA
rliu384@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 using different algorithms, evaluating their theoretical and experimental complexities on real world datasets.

CCS CONCEPTS

• **Mathematics of computing → Graph algorithms**.

KEYWORDS

MVC, Branch and Bound, Local Search, Approximation, Simulated Annealing, Hill Climbing

1 INTRODUCTION

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 using different algorithms, evaluating their theoretical and experimental complexities on real world datasets.

Objectives

- Implement an exact branch-and-bound algorithm
- Implement approximate algorithms that run in reasonable time and provide high-quality solutions with guarantees
- Implement 2 local search algorithms (without approximation guarantees)

Teamwork

- Branch-and-Bound, Local Search (HC) : Chujie Chen, Shuran Wen
- Local Search (SA), Approximation : Siying Cen, Ren Liu

2 PROBLEM DEFINITION

The MVC problem can be defined as following:

For a graph $G = (V, E)$, find a vertex subset $S \subseteq V$ so that for every edge $(u, v) \in E$, either $u \in S$ or $v \in S$ or both; In other words,every edge in E must have at least one vertex in S. The goal is to seek the vertex subset S with minimum cost, $|S|$.

*All authors contributed equally to this project.

3 RELATED WORK

There are many algorithms developed to solve MVC problem since last century. The most common algorithms include Branch-and-Bound (BnB) algorithm (BB), Greedy algorithm, Approximation algorithm, Genetic algorithm (GA) and Local Search (LS) algorithms. These algorithms can be categorized into two types: complete and incomplete algorithms. The complete algorithms (for example, BnB) guaranteed to find the optimum solution while the incomplete algorithms (e.g. Approximation, LS) not guaranteed the optimum. However, the incomplete algorithms are much more efficient than complete algorithms and they are sufficient for practical applications.

Though time complexity of BnB algorithm is usually exponential, it can be improved to $O(1.2738k + kn)$ by setting appropriate lower bound and upper bound [2]. The quality and time-complexity of LS algorithm can be balanced by combining heuristics algorithm [1].

4 ALGORITHMS

4.1 Branch and Bound

The problem is divided into n branches, then finds the minimum vertex cover from a VC called $C_i = V - \{v_i\}$ $(i = 1, 2, ..., n)$. The minimum vertex cover C_i is found by omitting removable vertices until it reach to the size k. If it's not found in all branches, for each pair of above vertex covers (VCs) of above vertex covers C_i, a VC called C_{ij} is created $(C_{ij} = C_i \bigcup C_j)$ and processed in the same way. The results of C_{ij} will be the minimum vertex covers of the problem.

The algorithm uses a parameter k as a bound as a bound value to limit the depth of search trees. It includes the advantages that:

- Search trees are smaller
- Search speed is improved

To determine value of k: every graph with n vertices and maximum vertex degree must have a minimum vertex cover of size at most have a minimum vertex cover of size at most $k = n - \lceil n/(\delta + 1) \rceil$. [3]

Besides, we have done plenty of optimization. Firstly, we use a HashMap to store the removable vertices number of each generated Vertex Cover graph. Therefore, if we need to get the removable vertices number of the same graph again, we don't need to re-caculate. Secondly, we generate a baseline graph at first, by keep removing the removable vertices in the graph following the ascendant order of each vertex's degree, and compare each algorithm output with this baseline. Moreover, the transfer order of removable vertices is also according to the ascendant order of vertex degree. We believe that the vertices with less degree will more likely be removed since they connect with fewer edges. The time complexity of our algorithm is reduced from $O(2^{|V|})$ to $O(n^8 + 2n^7 + n^6 + n^5 + n^4 + n^3 + n^2) \sim O(|V|^8)$ while the space complexity is $O(|E| + |V|)$.

The pseudo-codes can be found 1 and 2.

4.2 Local Search 1: Simulated Annealing

Simulated annealing (SA) is a probabilistic technique for approximating the global optimum of a given function. Specifically, it is a meta-heuristic to approximate global optimization in a large search space for an optimization problem.

Generally, there are 4 key conceptions involved:

- **The basic iteration:** At each step, the simulated annealing heuristic considers some neighboring state s_{new} of the current state s, and probabilisticly decides between moving the system to state s_{new} or staying in state s.
- **The neighbours of a state:** New states produced through Local Search.
- **Acceptance probabilities:** Generally if the objective of the task is F, the transition probability will be defined as a function of $P(F(s), F(s_{new}), T)$, or simplified as $P(s, s_{new}, T)$.
- **The annealing schedule:** The annealing schedule controls when to stop iteration. For every iteration k, the temperature T will be **cooled down**.When $k > k_{max}$ or $T < T_{min}$, the loop will stop.

Based on the great works done by Xu et al.[4], we defined our own SA system for Minimum Vertex Cover problem as Table 1.

SA algorithm is usually not very strict with what algorithm used to generate the initial state s_0. All SA schedules will follow a relatively fixed routine. The input of the routine includes $\{k_0, k_{max}, T_0, T_{min}, Seed, s_0\}$, which can be written as the pseudo-codes algorithm 3.

Same as other algorithms, we build the graph in $O(|E| + |V|)$ time. Degrees of each vertices are calculated in a total running time of $O(|E|)$. Sorting by degrees costs a running time of $O(|V|log|V|)$. For the SA scheduler we used in the experiment. set the default temperature as 1000000 with a cooling rate of 0.95, it can take a lot of time for the loop. Hence in addition to the total run time limit, we set a static loop time to check if the best state is not updated for a long time. If so, we regard the current time as **stable** and quit the loop. The limit of static time is set to $5s$ as default in our code. For every iteration in the loop, we have a $O(1)$ update function to choose a random vertex and flip its situation. Then we take at most $O(|E|)$ time to check if the flip disturb the vertex cover.

As for the space complexity, except from the graph, we use a HashMap to encode the vertex cover, which will take at most $O(|V|)$ space.

4.3 Local Search 2: Hill Climbing

Hill Climbing (HC) was implemented as one of the two local search algorithms. HC can run very fast and works well for a wide variety of practical problems while it sometimes get stuck at local maxima. We chose a normal greedy strategy here in the implementation of HC: in each step, move from the current search position to a neighboring position with better evaluation

Algorithm 1: Branch And Bound (partial)

```
Input: Graph $G$
Output: Vertex Cover $Rnt$
Function Main($G$):
    n = number of vertices of $G$;
    k = n - $\lceil n/(\Delta + 1) \rceil$;
    return BnB($G, k$);
;
Function BnB($G, k$):
    // Preprocess and Try Luck;
    for i from vertex with highest degree to lowest do
        $C_i = V - \{v_i\}$;
        $C_i$ = PreProcess (G, $C_i$);
        for i=1 to n-k do
            $C_i$ = ProcB (G, $C_i$, r);
            Update Rnt if encounter smaller VC;
        end
    end
    if Rnt is already lower bound then
        return Rnt;
    end
    //Part I
    for i from vertex with highest degree to lowest do
        $C_i = V - \{v_i\}$;
        $C_i$ = ProcA (G, $C_i$);
        for i=1 to n-k do
            $C_i$ = ProcB (G, $C_i$, r);
            Update Rnt if encounter smaller VC;
        end
    end
    // If reach Lower Bound -> skip Part II;
    if Rnt is already lower bound then
        return Rnt;
    end
    //Part II
    for each pair of $C_i, C_j$ found in Part I do
        $C_{ij}$ =$C_i \bigcup C_j$;
        $C_{ij}$ = ProcA (G, $C_{ij}$);
        for i=1 to n-k do
            $C_{ij}$ = ProcB (G, $C_{ij}$, r);
        end
    end
    Find Rnt from above $C_{ij}$;
    return Rnt;
;
Function PreProcess($G, C$):
    while the vertex cover C has removable vertices do
        C = C-{$v_{highestDegree}$};
    end
    return C
;
Function ProcA($G, C$):
    while the vertex cover C has removable vertices do
        for each removable vertex v of C do
            if Haven't consider $C - \{v\}$ before then
                r(C-{v}) = the number of removable
                  vertices of Vertex cover C-{v};
                $v_{max}$ = v where r(C-{v}) is a maximum;
                C = C-{$v_{max}$};
            end
        end
    end
    return C
;
```

Algorithm 2: Branch And Bound (cont.)

```
Input: Graph $G$
Output: Vertex Cover $Rnt$
Function ProcB($G, C, n$):
    for $i$ = 1 to $n$ do
        Find $v \in C$ that has exactly one neighbor $w \in C$;
        if $v$ is found then
            $C^{v,w} = (C \bigcup \{w\}) - \{v\}$ ;
            C = ProcA (G, $C^{v,w}$) ;
        end
    end
```

Minimum Vertex Cover SA System	
Objective	$F(s) = \|VertexCover(s)\|$
Neighbour	Randomly Flip a vertex v_i in/out VertexCover
Flag	If $v_i(s_{new}) \notin VC$, $flag(s_{new}) = 1$ else -1
Factor	$D(s, s_{new}) = 1 + flag(s_{new}) \cdot Degree(v_i(s))/\|E\|$
Probability	$P(s, s_{new}, T) = \exp(-(F(s_{new}) - F(s)) \cdot D(s, s_{new})/T)$
Cool Down	$T_{new} = T \cdot$ coolRate

Table 1: Customized SA System

Algorithm 3: General SA Routine

```
Result: Optimized Instance state $s$ searched by SA
initialization, let $s = s_0$, $T = T_0$;
for $k = 0; k < k_{max}$ and $T > T_{min}$ do
    $T$ = temperature($(k+1)/k_{max}$);
    Pick a random neighbour, $s_{new}$ = neighbour($s$);
    if $P(F(s), F(s_{new}), T) \geq$ random(0,1) then
        Update $s = s_{new}$;
    end
    return $s$;
end
```

function value. For our MVC problem, we have the original graph $G = (V, E)$.

- **Initial State:** A graph that is still a VC derived from G by removing an arbitrary vertex.
- **Evaluation Function:** Total degrees of all vertices in the current VC: $D(VC) = \sum_i d(v_i)$. The degree of a graph vertex v of a graph G is defined as the number of graph edges which touch v.
- **Highest-valued Successor:** We want to minimize the number of vertices at the same maximize the total degrees. Neighboring states are defined as states (VCs) that have one node smaller than the current state (VC). In other words, for candidate states with same degrees, the state with the highest degree is the next-best state.

The algorithm was implemented as described below. First we check if the original graph is a VC or not. We randomly delete a removable vertex to get the graph as the initial state. Here, removable vertices are nodes that can be removed resulting in a VC. In the each step of HC, we delete the removable vertex having the lowest degree. If a vertex with a lower degree is currently not removable, apparently it is not removable in future steps either. And thus, in the implementation, we can just iterate through all vertices in ascending order based on their degrees and delete the one that is removable at the current stage. The pseudo-code for this algorithm can be found at 4.

As discussed above, building the graph takes $O(|E|+|V|)$ time. Degrees of each vertices are calculated in a total running time of $O(|E|)$. Sorting by degrees costs a running time of $O(|V|log|V|)$. There are $O(|V|)$ steps, and each step we delete the removable vertex with the lowest degree. Although checking the removability of one vertex can cost a running time of $O(|V|)$ and deletion takes constant time, the whole time spent in those steps is amortized $O(|V|)$ as we only check removabilities of remaining vertices with early stops. So the overall time complexity is $O(|E|+|V|log|V|)$. And the space complexity is simply $O(|E|+|V|)$ to store the graph.

Algorithm 4: Local Search 2: Hill Climbing

```
Input: Graph $G(V, E)$, seed $s$
Output: Vertex Cover $VC$
Function HillClimbing($G, s$):
    // Initialize starting VC Randomly
    VC = MakeNode($G$, $s$);
    Sort vertices in VC by degrees in ascending order;
    while true do
        nextVC = findSuccessor($VC$);
        if nextVC cannot be obtained then
            return VC;
        end
        VC = nextVC;
    end
    return VC;
;
Function MakeNode($G$, $s$):
    while true do
        Randomly choose a vertex v based on s from G;
        if v is removable then
            return G - v;
        end
    end
;
Function findSuccessor($VC$):
    while There are remaining unexplored vertices do
        if Current vertex is removable then
            return VC - curr;
        end
    end
    return null;
```

4.4 Approximation

Approximation is an algorithm for quickly finding a solution that is provably not very bad. It does not guarantee the optimal solution, however, approximation provides a bound on the quality of the solution.

Based on the proof introduced in CSE 6140 lecture, we defined our 2-Approximation algorithm for Minimum Vertex Cover problem as following:

Algorithm 5: 2-Approximation algorithm

```
Result: Optimized solution $C$ searched by Approximation
initialization, let $C = \emptyset$, edge set $E' = E$;
while $E' \neq \emptyset$ do
    Randomly select an edge $e = (u, v)$ in $E'$;
    $C = C \cup \{u, v\}$;
    for each edge $e = (s, t)$ in $E'$ do
        if $s \in \{u, v\}$ or $t \in \{u, v\}$ then
            $E' = E' - (s, t)$
        end
    end
    return $C$;
end
```

This algorithm can be proved to be a 2-Approximation algorithm, which means that
$OPT(MVC) \leq Approximation(MVC) \leq 2OPT(MVC)$.

Time and Space complexity: The running time of approximation is polynomial; in our 2-Approximation algorithm, the worst-case time complexity is $O(E^2)$.
(O(E) for randomly select one edge e in E' each time, O(E) for scanning all edges in E' to remove edges which share at least one common vertex with e.)
The space complexity is $O(V + E)$.

5 EMPIRICAL EVALUATION

All these four algorithms are implemented by Java, with least version of Java 8. No other open-source libraries are used. All the results for a random seed based algorithms are calculated by the mean performance of 10 independent tests with randomized test seeds. For simulated annealing, we set the **static time** limit as 10 seconds, which means no update within the latest 10 minutes then the algorithm will quit.

5.1 Branch and Bound

(1) **Environment Parameters:**
   - Time Limit: 10 min = 600s
(2) **Performance:** See at Table 3

5.2 Local Search: Simulated Annealing (SA)

(1) **Environment Parameters:**
   - Seed: Randomized 50 random seeds, in the table is the mean result.
   - Time Limit: 60 sec
   - CPU: Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz
   - RAM: 16 G
   - OS: Ubuntu 20.04.1 LTS
   - Cool Rate: 0.95
   - Static Quit Time: 10 sec
(2) **Performance:** Mean performance of 10 independent tests at Table 4. The table shows that generally, our simulated annealing algorithm attains great results among most graphs. Most relative error rates are lower than 10%, some are even stably lower than 5%. However, due to the randomized seed and limited running time, some results sees to be very various. Some specific graph, like *power.graph* and *star.graph*, the algorithm performs not as good as other graphs, like *hep-th.graph* and *as-22july06.graph*, which might because the differences in the graph's unique structures.
(3) Qualified Runtime Distributions (QRTDs) Plot Examples

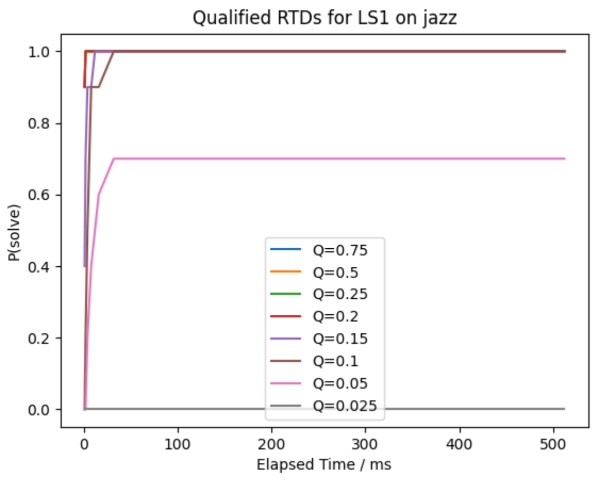

Figure 1: QRTDs for *jazz.graph* by SA

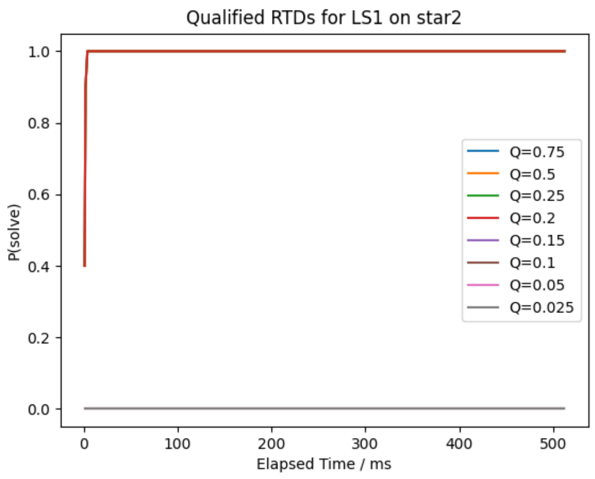

Figure 2: QRTDs for *star2.graph* by SA

Figure 1 and 2 are the QRTDs plots for *jazz.graph* and *star2.graph* by SA algorithm, which are chosen as examples of normal and massive graphs. From the graph we see when the quality is larger than 0.05, the algorithm cannot get solved at all times. Another feature is that all the curve rise rapidly only at the beginning, which reveals that our SA have relatively weak ability to adjust solution when later.

(4) Solution Quality Distributions (SQDs) Plot Examples

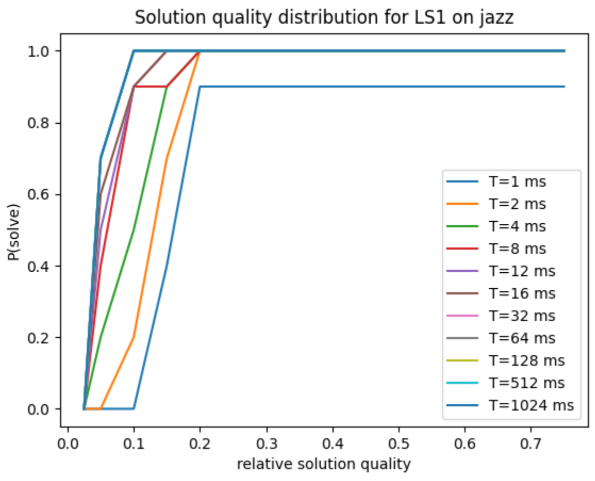

Figure 3: QRTDs for *jazz.graph* by SA

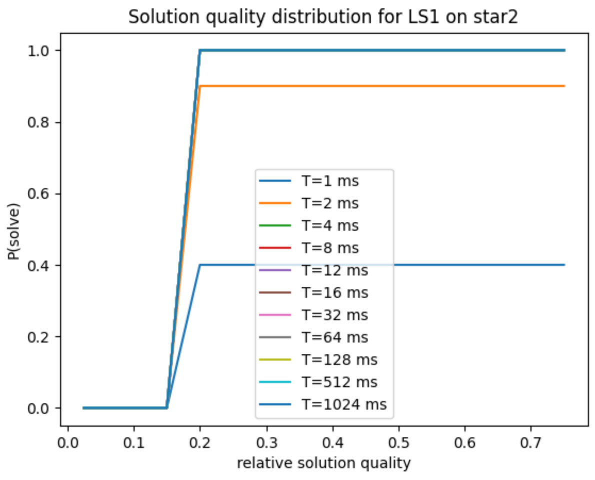

Figure 4: QRTDs for *jazz.graph* by SA

Similarly, SQDs are plotted the same way with qualities as x-axis while ratios for solving the problem serves as y-axis. Figure 3 and 4 are the SQDs plots for *jazz.graph* and *star2.graph* by SA algorithm. We see clearly that for both graphs, the solving probability increases as the time becomes longer. For smaller graphs, time impact might be more apparent at the beginning phase. While for massive graphs, the impact by time to the solving probability may be divided in to several discrete levels. Since the run time is quite limited in our test, there should be better results if we spend more time for SA to search the optimal solution.

5.3 Local Search 2: Hill Climbing (HC)

(1) **Environment Parameters:**
- Seed: Randomized 50 random seeds, in the table is the mean result.
- Time Limit: 60 sec
- CPU: Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz
- RAM: 16 G
- OS: Ubuntu 20.04.1 LTS

(2) **Performance:** Mean performance of 10 independent tests at Table 5. The table shows that generally, our hill climbing algorithm attains very good results among all graphs. Most relative error rates are lower than 5%, some are even stably lower than 3%. Besides, the algorithm can run very fast. Some specific graph, like *dummy2.graph* and *star2.graph*, the algorithm performs not as good as other graphs, like *hep-th.graph* and *as-22july06.graph*, which might because the differences in the graph's unique structures and the limitation of hill climbing that can only converge to one local peak.

(3) QRTDs Plot Examples

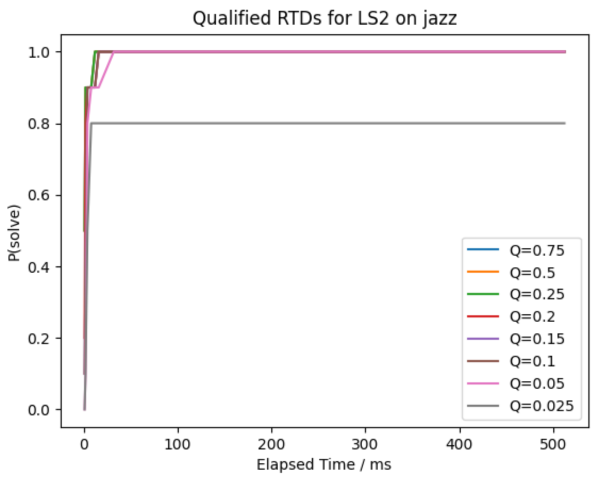

Figure 5: QRTDs for *jazz.graph* by HC

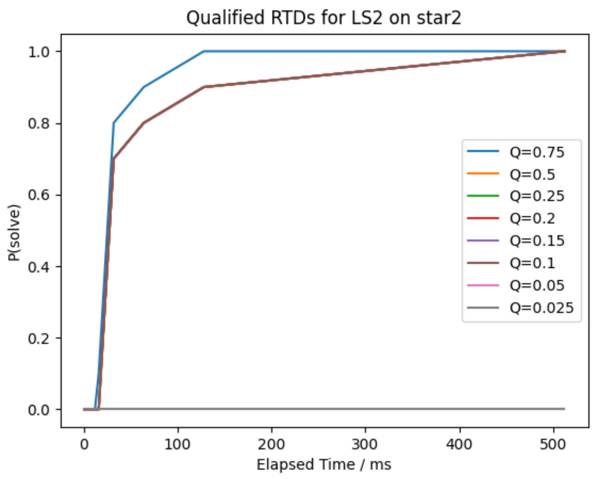

Figure 6: QRTDs for *star2.graph* by HC

Figure 5 and 6 are the QRTDs plots for *jazz.graph* and *star2.graph* by HC algorithm. From the graph we see when the quality is larger than 0.025, the algorithm cannot get solved at all times. Similar to SA, all the curve rise rapidly only at the beginning, which reveals that our HC algorithm have a great fast speed.

(4) SQDs Plot Examples

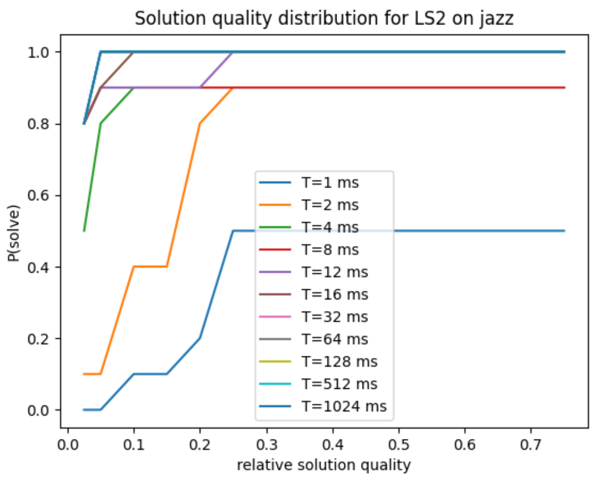

Figure 7: SQDs for *jazz.graph* by HC

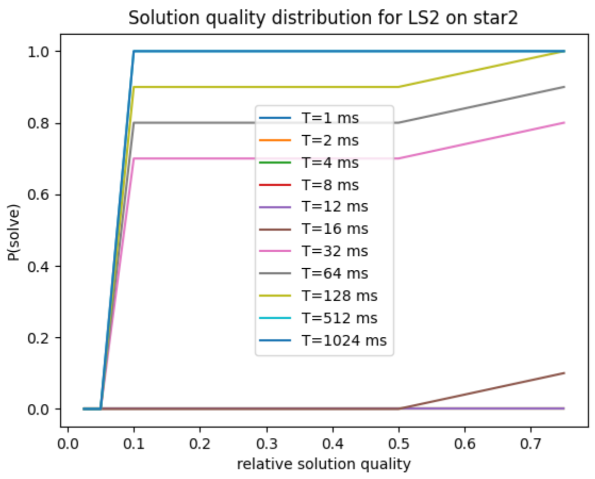

Figure 8: SQDs for *star2.graph* by HC

Then, Figure 7 and 8 are the SQDs plots for *jazz.graph* and *star2.graph* by HC algorithm. We also learn clearly that for both graphs, the solving probability increases as the time becomes longer. For smaller tasks, it seems that we cannot obtain better result when given more time. But there is a remarkable increase on the quality when the graph is large.

5.4 Approximation

(1) **Environment Parameters:**
- Seed: 0 - 9, in the table is the mean result of 10 runs for each graph.
- Time Limit: 600 sec
- CPU: Intel(R) Core(TM) i7-7500 CPU @ 2.70GHz
- RAM: 12 G
- OS: Window 10

(2) **Performance:** See at Table 6.

(3) **Quality Bound**: From result at Table 6, we can see that except for graph "as-22july06", all the approximation results for MVC are within 2OPT, which is consistent to the proof that the approximation algorithm is a 2 bound algorithm.

6 DISCUSSION

6.1 Comparison between Hill Climbing and Simulated Annealing

From the results of QRTDs and SQDs plots, clear comparison can be made between the two Local Search algorithms: Simulated Annealing and Hill Climbing. For example, in the figures on *jazz.graph*, Hill Climbing algorithm are more likely to reach the optimal, but Simulated Annealing are faster to converge. For the *start2.graph*, SA is still faster but with lower accuracy.

From an overall scope, we can see in the Figure 9 and find out all graph's relative error by these two algorithms. While generally, Hill Climbing algorithm has the lower error rate, the Simulated Annealing may still have better performances for some specific tasks.

Another factor that would impact the results is the cutoff time. As is shown in the box plots 10 and 11, Simulated Annealing algorithm's results are more variant, which may because the running time is too limited for the algorithm to reach its best score.

6.2 Comparison between all methods

(1) **Branch and Bound:**
Branch and Bound algorithm is the most precise method among all the 4 algorithms solving MVC problem. It performs very well when solving the small graph, with the 0.00% error rate; when performing larger graphs, the error rate increase but it is still acceptable (within 6.73%). However, the running time of BnB algorithm vary a lot, some graphs take hundreds of seconds for calculation while some graphs can be calculated within 1 second. Thus, the vertex degrees and bound selection influence the BnB efficiency, rather than graph's size.

(2) **Local Search:**
Although optimal solutions are not guaranteed, the two local search algorithms (Simulating Annealing and Hill Climbing) are efficient and stable. Acceptable relative errors in the range of 0.00% and 9.28% are achieved by both algorithms. The Hill Climbing algorithm shows similar results and relative errors as Simulating Annealing, but the Hill Climbing algorithm runs much faster than SA algorithm.

(3) **Approximation:**
The relative error of Approximation algorithm is very

large compared to other algorithms, however, it guarantees 2 OPT quality while the running time is faster than SA algorithm for most of the graphs. In the future we can improve it by greedy strategy

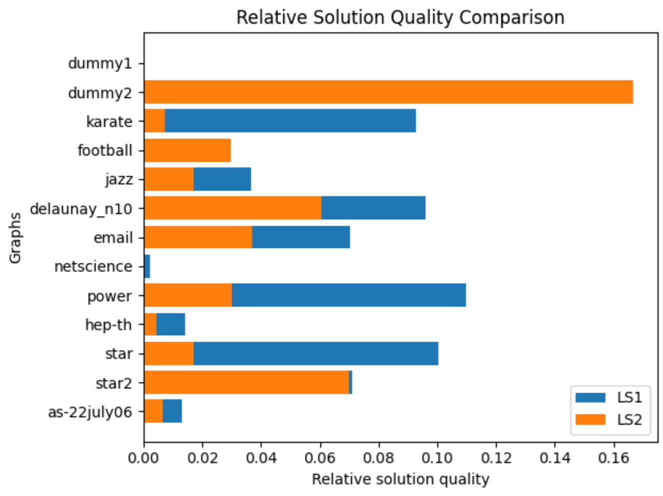

Figure 9: Performance comparison between SA and HC

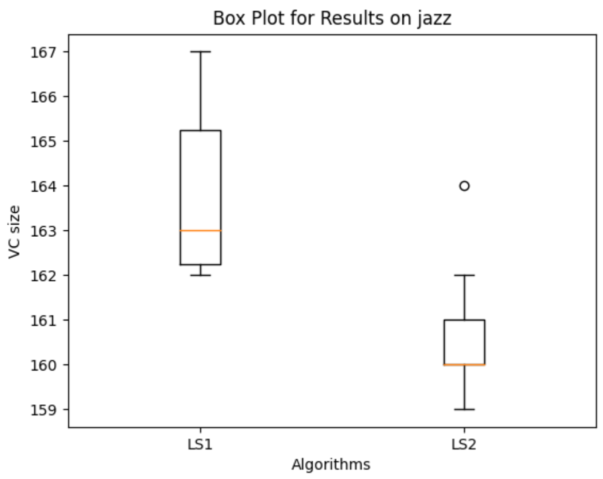

Figure 10: Boxplot for *jazz.graph* by SA and HC

7 CONCLUSION

The BnB method ensures the best quality of MVC results with the minimum relative error, however the running time is the longest. Two local search methods balance between quality and time, especially the Hill Climbing method which finishes calculating in 1 sec while maintaining good quality. The approximation method is also quick and controls the quality of results within 2OPT. However, the relative error of approximation is the highest among all methods.

Overall, two Local Search methods stand out in all the methods.

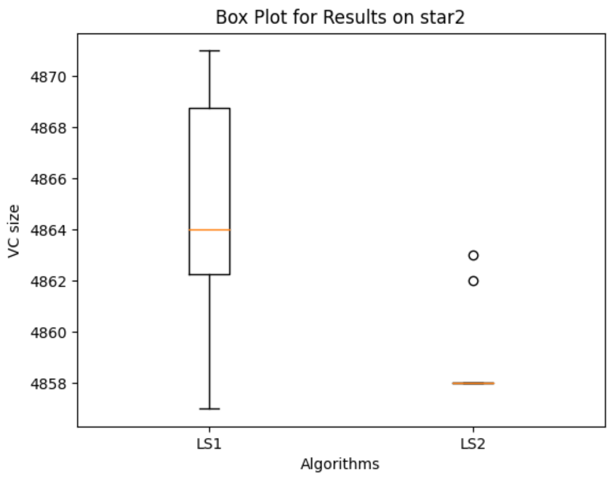

Figure 11: Boxplot for *star2.graph* by SA and HC

All results can be found in following pages.

REFERENCES

[1] Shaowei Cai. Balance between complexity and quality: Local search for minimum vertex cover in massive graphs. 2015.
[2] Jianer Chen, Iyad A Kanj, and Ge Xia. Improved parameterized upper bounds for vertex cover. pages 238–249, 2006.
[3] Alexander K Hartmann and Martin Weigt. Statistical mechanics of the vertex-cover problem. *Journal of Physics A: Mathematical and General*, 36(43):11069–11093, oct 2003.
[4] Xinshun Xu and Jun Ma. Letter: An efficient simulated annealing algorithm for the minimum vertex cover problem. *Neurocomput.*, 69(7–9):913–916, March 2006.

Comprehensive Table								
Instance.graph	|V|	|E|	Opt		BnB	LS1	LS2	Approx
dummy1	6	6	2	Time(s)	0.005	0.001	0.001	0.001
				VC Value	2	2.0	2.0	4
				RelErr	0.0000	0.0000	0.0000	1.0000
dummy2	6	7	3	Time(s)	0.010	0.001	0.001	0.000
				VC Value	3	3.0	3.5	5
				RelErr	0.0000	0.0000	0.1667	0.6667
karate	34	78	14	Time(s)	0.020	0.003	0.004	0.002
				VC Value	14	15.3	14.1	25
				RelErr	0.0000	0.0928	0.0071	0.7857
football	115	613	94	Time(s)	0.335	0.006	0.003	0.010
				VC Value	94	96.3	96.8	114
				RelErr	0.0000	0.0245	0.0298	0.2128
jazz	198	2742	158	Time(s)	486.297	0.011	0.004	0.071
				VC Value	158	163.8	160.7	191
				RelErr	0.0000	0.0367	0.0171	0.2089
delaunay_10	1024	3056	703	Time(s)	73.589	0.241	0.011	0.066
				VC Value	743	770.6	745.6	1006
				RelErr	0.0569	0.0961	0.0606	0.4310
email	1133	5451	594	Time(s)	192.40	0.202	0.011	0.128
				VC Value	609	635.8	616.0	934
				RelErr	0.0253	0.0703	0.0370	0.5724
netscience	1589	2742	899	Time(s)	1.335	0.408	0.012	0.055
				VC Value	899	901.1	899.0	1339
				RelErr	899	0.0023	0.0000	0.4894
power	4941	6594	2203	Time(s)	45.298	4.751	0.022	0.271
				VC Value	2270	2444.9	2269.6	4254
				RelErr	0.0304	0.1098	0.0302	0.9310
hep-th	8361	15751	3926	Time(s)	24.357	15.135	0.030	0.959
				VC Value	3942	3981.3	3942.7	6588
				RelErr	0.0041	0.0141	0.0043	0.6780
star	11023	62184	6902	Time(s)	83.766	25.026	0.033	12.12
				VC Value	7017	7593.8	7020.6	10902
				RelErr	0.0167	0.1002	0.0172	0.5795
star2	14109	98224	4542	Time(s)	226.55	34.398	0.046	29.947
				VC Value	4848	4864.7	4858.9	7932
				RelErr	0.0673	0.0710	0.0698	0.7464
as-22july06	22963	48436	3303	Time(s)	276.42	57.957	0.061	6.101
				VC Value	3326	3345.6	3324.4	7371
				RelErr	0.0070	0.0129	0.0065	1.2316

Table 2: All Results for Four Algorithms

Minimum Vertex Cover Test						
Graph Name	$\|V\|$	$\|E\|$	Opt	Result	RelErr	time(s)
dummy1.graph	6	6	2	2	0.0000	0.005
dummy2.graph	6	7	3	3	0.0000	0.010
karate.graph	34	78	14	14	0.0000	0.020
football.graph	115	613	94	94	0.0000	0.335
jazz.graph	198	2742	158	158	0.0000	486.297
delaunay_n10.graph	1024	3056	703	743	0.0569	73.589
email.graph	1133	5451	594	609	0.0253	192.395
netscience.graph	1589	2742	899	899	0.0000	1.335
power.graph	4941	6594	2203	2270	0.0304	45.298
hep-th.graph	8361	15751	3926	3942	0.0041	89.701
star.graph	11023	62184	6902	7017	0.0167	83.766
star2.graph	14109	98224	4542	4848	0.0673	226.552
as-22july06.graph	22963	48436	3303	3326	0.0070	276.415

Table 3: Test Results for Branch and Bound Algorithm

Minimum Vertex Cover Test						
Graph Name	$\|V\|$	$\|E\|$	Opt	Result	RelErr	time(s)
dummy1.graph	6	6	2	2.0	0.0000	0.001
dummy2.graph	6	7	3	3.0	0.0000	0.001
karate.graph	34	78	14	15.3	0.0928	0.003
football.graph	115	613	94	96.3	0.0245	0.006
jazz.graph	198	2742	158	163.8	0.0367	0.011
delaunay_n10.graph	1024	3056	703	770.6	0.0961	0.241
email.graph	1133	5451	594	635.8	0.0703	0.202
netscience.graph	1589	2742	899	901.1	0.0023	0.408
power.graph	4941	6594	2203	2444.9	0.1098	4.751
hep-th.graph	8361	15751	3926	3981.3	0.0141	15.135
star.graph	11023	62184	6902	7593.8	0.1002	25.026
star2.graph	14109	98224	4542	4864.7	0.0710	34.398
as-22july06.graph	22963	48436	3303	3345.6	0.0129	57.957

Table 4: Test Results for Local Search Algorithm 1: Simulated Annealing

Minimum Vertex Cover Test						
Graph Name	$\|V\|$	$\|E\|$	Opt	Result	RelErr	time(s)
dummy1.graph	6	6	2	2.0	0.0000	0.001
dummy2.graph	6	7	3	3.5	0.1667	0.001
karate.graph	34	78	14	14.1	0.0071	0.004
football.graph	115	613	94	96.8	0.0298	0.003
jazz.graph	198	2742	158	160.7	0.0171	0.004
delaunay_n10.graph	1024	3056	703	745.6	0.0606	0.011
email.graph	1133	5451	594	616.0	0.0370	0.011
netscience.graph	1589	2742	899	899.0	0.0000	0.012
power.graph	4941	6594	2203	2269.6	0.0302	0.022
hep-th.graph	8361	15751	3942	3942.7	0.0043	0.030
star.graph	11023	62184	6902	7020.6	0.0172	0.033
star2.graph	14109	98224	4542	4858.9	0.0698	0.046
as-22july06.graph	22963	48436	3303	3324.4	0.0065	0.061

Table 5: Test Results for Local Search Algorithm 2: Hill Climbing

Minimum Vertex Cover Test						
Graph Name	$\lvert V \rvert$	$\lvert E \rvert$	Opt	Result	RelErr	time(s)
dummy1.graph	6	6	2	4	1.0000	0.001
dummy2.graph	6	7	3	5	0.6667	0.000
karate.graph	34	78	14	25	0.7857	0.002
football.graph	115	613	94	114	0.2128s	0.010
jazz.graph	198	2742	158	191	0.2089	0.0712
delaunay_n10.graph	1024	3056	703	1006	0.4310	0.066
email.graph	1133	5451	594	934	0.5724	0.128
netscience.graph	1589	2742	899	1339	0.4894	0.055
power.graph	4941	6594	2203	4254	0.9310	0.271
hep-th.graph	8361	15751	3926	6588	0.6780	0.959
star.graph	11023	62184	6902	10902	0.5795	12.120
star2.graph	14109	98224	4542	7932	0.7464	29.947
as-22july06.graph	22963	48436	3303	7371	1.2316	6.101

Table 6: Test Results for 2-Approximation Algorithm