Minimum Vertex Cover Project Report

JIAWEI CHEN, QINYU WANG, YIFENG WANG, and ZIJIAN LI

The Minimum Vertex Cover problem is a member of NP-complete problem which has a broad applications in computational complexities. In this project, we utilize different algorithms including branch and bound, heuristics with approximation guarantees, and local search (simulated annealing and hill climbing) to tackle this problem. The solution quality of each algorithm is evaluated with their theoretical and experimental complexities being analyzed, which provide implications for their applications in real word datasets.

Additional Key Words and Phrases: minimum vertex cover, branch and bound, approximation, local search, simulated annealing, hill climbing, quality, relative error

ACM Reference Format:
Jiawei Chen, Qinyu Wang, Yifeng Wang, and Zijian Li. 2020. Minimum Vertex Cover Project Report. *ACM Trans. Graph.* , (November 2020), 10 pages.

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. Albeit its significance, the search for optimal solution for the problem is not viable due to its intractable nature. In this project, we explore four algorithms: branch and bound (BnB), heuristics with approximation guarantees (HA), simulated annealing (SA), and hill climbing (HC). The last two algorithms are belonged to local search with no guarantees. Each algorithm is evaluated for its solution quality and run time within the cutoff criterion. The theoretical and experimental complexities of different algorithms are analyzed and compared for the purpose of searching for a best near-optimized solution.

The contributions of each group member are summarized below. Qinyu is charge of the implementation of branch and bound. Yifeng works on the construction of heuristics with approximation guarantess. JiaWei takes the lead to realize the simulated annealing and Zijian explores the implementation of hill climbing. All group members take part in the graph parsing, experiments running, results analyzing and report writing. All group members have contributed similar amount of efforts.

Authors' address: Jiawei Chen, jchen897@gatech.edu; Qinyu Wang, qwang437@gatech.edu; Yifeng Wang, ywang3627@gatech.edu; Zijian Li, zjli415@gatech.edu.

0730-0301/2020/11-ART $15.00
https://doi.org/

2 PROBLEM DEFINITION

A vertex cover of a graph is a set of vertices that includes at least one endpoint of every edge of the graph (https://en.wikipedia.org/wiki/Vertex_cover#cite_note-7). The problem of finding a minimum vertex cover (MVC) is a classical NP-complete problem which can not be solved with a solution in polynomial time. However, several algorithms are able to approximate the near-optimal solutions within polynomial time.

Mathematically, the MVC problem is defined as follows. Given an undirected graph $G = (V, E)$ with a set of vertices V and a set of edges E, a vertex cover is a subset $C \subseteq V$ such that $\forall(u, v) \in E : u \in C \vee v \in C$. The Minimum Vertex Cover problem is therefore simply the problem of finding minimizing $|C|$. The following Figure 1 shows examples of minimum vertex cover.

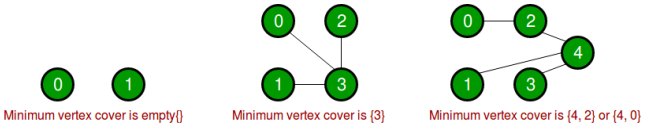

Fig. 1. Examples of minimum vertex cover (https://www.geeksforgeeks.org/vertex-cover-problem-set-1-introduction-approximate-algorithm-2/).

3 RELATED WORK

Extensive efforts have been made to search the approximate polynomial-time algorithms to solve the MVC problem. For instance, the factor-2 heuristic approximation algorithm was discovered to run in polynomial time and is able to generate the lower bound of the optimal solution [Garey and Johnson 1979] [Papadimitriou and Steiglitz 1998]. Since the results derived from the approximate algorithm is no more than twice the size of optimal solution. Some other approximation algorithms can achieve a slightly better approximation factor, such as an approximation factor of $2 - \Theta(1/\sqrt{\log |V|})$ [Karakostas 2005] or an approximation factor of $2/(1+\delta)$ [Karpinski and Zelikovsky 1996]. Some other work prove that if the graph type is either bipartite graph (König's theorem) or tree graph, the MVC problem can be solved in polynomial time.

An efficient simulated annealing algorithm is proposed by [Xu and Ma 2006] to solve the MVC problem. A novel acceptance function is defined for every vertex, which proves to be effective in finding the near-optimal solutions with a higher probability. This acceptance function is referred in our implementation of simulated annealing algorithm.

4 DATA

In this project, a total of eleven graph instances are tested with our implemented algorithms to evaluate their performances in solving MVC problem. All these benchmark instances are real and random datasets extracted from the 10th DIMACS challenge and are represented in Metis I/O format. The numbers of vertices, edges, and the

Table 1. The vertex count, edge count, and optimal solution of MVC for each benchmark graph instance.

Dataset	|V|	|E|	Opt
jazz	198	2742	158
karate	34	78	14
football	115	613	94
as-22july06	22963	48436	3303
hep-th	8361	15751	3926
star	11023	62184	6902
star2	14109	98224	4542
netscience	1589	2742	899
email	1133	5451	594
delaunay n10	1024	3056	703
power	4941	6594	2203

Table 2. The minimum vertex cover (VC) value and VC list of heuristic approximation and hill climbing for dummy1.graph and dummy2.graph.

	Heuristic Approximation		Hill Climbing	
Dataset	VC Value	VC list	VC Value	VC list
dummy1	2	1, 2	2	3, 1
dummy2	6	1, 2, 3, 4, 5, 6	3	2, 3, 4

optimal solution for minimum vertex cover for each graph instance are summarized in the Table 1. The description for each graph is omitted here.

In addition to the graphs shown above, we run our implemented heuristic approximation and hill climbing algorithms on two small graphs, *dummy1* and *dummy2* to test the graph parsing function and the validity of outputs. The test results are shown in the Table 2.

5 ALGORITHMS

5.1 Branch and Bound

The basic idea of Branch-and-Bound algorithm is to investigate smaller subset graph of the parent graph, G, for optimal vertex cover solutions. We evaluate the subset graph at every level of the tree and prune the branches according to upper and lower bounds. In the current implementation, we initially set the upper bound as the best initial solution by 2-approximate vertex cover, and updated it if fewer nodes are needed in a soluble vertex cover. The initial assignment of the upperbound is essentially important because the in-depth search performs noticeably slow in relatively large graph. Our lower bound is set as the summation of the current vertex cover and the approximate vertex cover of the remaining graph. We started the depth-first search with vertex in ascending order of vertex degree. If current vertex is not in current vertex cover, then all its neighbors are removed from remaining vertex list and remaining graph. Within the loop, if there's no edges left, return the vertex cover and update the upperbound; if the solution isn't generated yet, we retrieve the vertex with highest degree from the remaining graph and keep exploring the subproblems. The search will terminate when lowerbound is no more than the upperbound.

Since the Branch-and-bound algorithm expands two subproblems (either adding the current highest degree vertex to the vertex cover or adding the neighbors of the current highest degree vertex to the vertex cover), the time complexity of this algorithm can be as much as 2^n (exponential scale). In our implementation, the most space-consuming data structure is a stack to store the pair of (current vertex cover, the remaining vertices, the remaining graph), the worst-case space complexity is $(O(|V|) + O(|E|)) * O(d)$, where $O(d)$ is the depth of the complete BnB search tree.

The pseudo code of implementing branch-and-bound algorithm (Algorithm 1) is attached below.

Algorithm 1: Branch-and-bound algorithm

```
Data: Graph G, cutoff time
Result: minimum vertex cover
improvedSolution <- [];
improvedSolution add (curTime, |G.nodes|);
VC <- [];
VIncluded <- [];
GRemain <-G;
VRemain <- vertices in G by ascending degree;
UpperBound = approx(G,weight=None);
stack.push(VIncluded, GRemain, VRemain);
while stack not empty AND runTime < cutoff time do
    VIncluded, GRemain, VRemian = stack.pop();
    LowerBound = |VC|+|maxmatching of GRemain|;
    if no edges in GRemain then
        if |VIncluded| < UpperBound then
            UpperBound = |VIncluded|;
            VC = VIncluded;
            improvedSolution add (curTime, |VC|);
        end
    end
    else
        if |VRemain !=0| then
            v, vdegree = VRemain.pop();
        end
        if LowerBound <= UpperBound then
            VI = VI + GRemain.neighbors(v);
            VR = GRemain.neighbors(v).remove();
            GR = GRemain;
            GR.removenodesfrom(GRemain.neighbors(v));
            stack.put(VI, VR, GR);
            GRemain.remove(v);
            VIncluded.add(v);
            stack.put(VIncluded, VRemain, GRemain)
        end
    end
end
return |VC|, VC, improvedSolution
```

5.2 Heuristics with Approximation Guarantees

Given an undirected graph $G = (V, E)$, where $|V| = n1$ and $|E| = n2$. There are two empty sets, Set A and Set B. We arbitrarily pick an edge i from the Set E and find all edges that are adjacent to either nodes on the edge i. Then we remove all those edges including the edge i from the Set E to the edge B. At the same time, we add two nodes on edge i to the Set A. We repeat this step until the Set E becomes an empty set. The Set A is the solution and $|A|$ is the size of this Vertex Cover Set, where $|A| = 2|B|$.

This approximation algorithm can be used to set a lower bound for the Minimum Vertex Cover problem. We can view this algorithm in a different way. All edges in the Set B are not adjacent. For each edge in the Set B, there must exist at least one end or node in the Minimum Vertex Cover set, Set M. Therefore, we can get:

$$|B| \leq |M|$$

For the Set B, we have the corresponding Set A that containing all nodes in edges in Set A. We also know this algorithm cannot guarantee the optimal solution. So, we can get:

$$|A| = 2|B|$$

$$|M| \leq |A|$$

$$|M| \leq |A| \leq 2|M|$$

$$|A|/2 \leq |M| \leq |A|$$

Based on the above, we can prove that this algorithm is 2-approximation algorithm and the lower bound for the Minimum Vertex Cover problem is $|A|/2$.

Here is the pseudo code of the heuristic approximation (Algorithm 2)

Algorithm 2: Heuristic approximation

```
Data: graph G, cutoff time, random seed
Result: minimum vertex cover
vertexCover <- empty set;
improvedSolution <- [];
uncoveredEdges <- edge set E of G;
initialize random with random seed;
while uncoveredEdges != [] AND runTime < cutoff time do
    u, v = two endpoints of a randomly selected edge in
      uncoveredEdges;
    vertexCover add u and v;
    update the uncoveredEdges by removing any edge with
      with either u or v as the endpoint;
end
improvedSolution add (curTime, vertexCover size);
return vertexCover size, vertexCover, improvedSolution
```

The in-theory time and space complexity of heuristic approximation is analyzed below. The while loop takes $O(|E|)$ time to traverse each edge in the evolving *uncoveredEdge* list that is initialized with the edge set E. The update *uncoveredEdge* step takes $O(|E|)$ time to find and remove the edges with endpoints just added in the covered in current vertex cover set. Overall, the time complexity is $O(|E|^2)$. Since three lists are used to keep track of current vertex cover, uncovered edges, and the improved solution, the overall space complexity is $max(O(|V|, O|E|))$.

5.3 Hill Climbing

The logic of this algorithm is to initialize the vertex cover set with the full vertex set of the graph and gradually remove vertex out of this set whiling keeping it validity until every vertex has been checked. The resulted vertex cover set is the near-optimal solution (MVC) achieved by hill climbing algorithm.

At first, sort the vertices of the graph by the ascending order of the degree. The degree of a vertex is defined by the number of edges that have one of the endpoint as this vertex. The MVC set is initially assigned with the full vertex set of the graph. Since the vertices of every edge of the graph are contained in this set, the initial MVC set is a valid vertex cover. The next step is to randomly select a vertex with the least degree as a candidate vertex to be removed from the vertex set. The reason to pick the vertex with the smallest degree to remove is that there is less possible that the picked vertex is an "essential" vertex and upon removal would result in a invalid vertex set. That is, the most "essential" vertices (with larger degrees) always rank last to be selected to be removed. For each potential removal candidate, the validity of the MVC set after removing this vertex is checked. If the MVC set is still valid, it is safe to remove this picked vertex, otherwise add this vertex back to the MVC set. To be noted, each vertex will only be checked once. If a vertex is checked to be "essential", it will be still essential when the MVC set size is smaller. The reason is that a vertex is marked as "essential" only when one of the vertex ends of its edge collection has already been moved out of the MVC set. If this vertex is removed as well, at least one of the edge is not covered by the MVC set and breaks the validity. Repeat the above process until the every vertex has been checked. During each iteration, if an improved solution is generated, it would be recorded by *improvedSolution* together with the current time. The best MVC solution together with *improvedSolution* set are returned.

The pseudo code of implementing hill climbing algorithm (Algorithm 3) and the helper function to check the validity of the current MVC set (Algorithm 4) are attached below.

The in-theory time and space complexity of the hill climbing algorithm is analyzed below. Sort the vertices by the degree takes $O(\log |V|)$ time. The outter and inner while loop to iteratively check each vertex takes $O(|V|)$ time. The removal of a vertex from the MVC set or sorted vertex list takes $O(|V|)$ time. To check whether the current MVC set is valid takes $O(|E|)$ time. Overall, the time complexity of the hill climbing algorithm is $O(|V|*(max(|V|+|E|)))$. Since three lists are used to store the current vertex cover, best vertex cover, and improved solutions, respectively, the space complexity is $O(|V|)$.

5.4 Simulated Annealing

The basic idea of simulated annealing is to model the physical process of heating a material and then slowly lowering the temperature to minimize the system energy. In this method, a temperature variable is used to bound the iteration times and the probability to

Algorithm 3: Hill climbing

```
Data: graph G, cutoff time, random seed
Result: minimum vertex cover
vertexCover <- V;
improvedSolution <- [];
bestVC = vertexCover;
improvedSolution add (curTime, bestVC);
degreeSorted <- list of nodes in G sorted by ascending
  degree;
initialize random with random seed;
while degreeSorted != [] AND runTime < cutoff time do
    leastDegreeVertices <- list of smallest degree nodes in
      degreeSorted;
    while leastDegreeVertices != [] AND runTime < cutoff time
     do
        curVertex = randomly selected node in
          leastDegreeVertices;
        vertexCover remove curVertex;
        if remaining vertexCover not valid then
            vertexCover add curVertex
        end
        leastDegreeVertices remove curVertex;
        degreeSorted remove curVertex;
        if vertexCover size < bestVC size then
            improvedSolution add (curTime, vertexCover
              size);
            bestVC = vertexCover
        end
    end
end
return bestVC size, bestVC, improvedSolution
```

Algorithm 4: Check validity of vertex cover

```
Data: vertexCover, removedVertex, graph G
Result: validity of the vc set
curNeighbors <- list of neighbors of removedVertex in G;
for each neighbor in curNeighbors do
    if neighbor not in vertex cover set then
        return False
    end
end
return True
```

accept a "worsening" neighbor solution. In the previous algorithm Hill Climbing, it may delete a vertex in every iteration and it is highly possibly the delete vertices in the early stage result in the solution to be stuck in a local minimum. The Simulated Annealing algorithm, however, may solve this problem. When the temperature is high, the algorithm would accept a neighboring solution which is worse than the current solution which may help the solution to jump out of the local minimum. The basic algorithm contains the following steps. Firstly, set the initial temperature, end temperature, and the temperature folding ratio. As long as the current temperature is higher than the end temperature and the lag time not exceeding the cutoff threshold, the while loop iterates. In each iteration, the temperature decreases in the scale of the folding ratio. Then, a vertex is randomly chosen from the original vertex cover. There are two cases. In one scenario, if the random chosen vertex is in current vertex cover, delete this vertex and check whether the rest vertex cover is valid. If it is not valid, add this vertex back to the vertex cover. Otherwise, a new improved solution is generated. In another scenario, the chosen vertex is not in the current vertex cover. Add this randomly selected vertex to the current vertex cover and calculate the probability p (a function of vertex degree and current temperature) of accepting this vertex. If p is larger than a randomly generated value between 0 and 1, accept this vertex, otherwise remove it from the current vertex cover. Hence, we may accept a worse solution (adding back the vertices deleted in the previous steps of the Hill Climbing) based on the probability p in simulated Annealing, which may help the solution to jump out of the local minimum. During each iteration, if an improved solution is generated, it would be recorded by *improvedSolution* together with the current time. The best MVC solution together with *improvedSolution* set are returned.

Here is the pseudo code (Algorithm 5):

The in-theory time and space complexity of the simulated annealing algorithm is analyzed below. The time complexity of the while loop is primarily governed by the initial and end temperature, and the temperature folding ratio (denote a time complexity of $O(T)$). Within the while loop, the step of removing a vertex from the current vertex cover takes $O(|V|)$ time and check the validity of the current vertex cover takes $O(|E|)$ time. Overall, the time complexity of the hill climbing algorithm is $O(|T| * (max(|V| + |E|)))$. Since three lists are used to store the current vertex cover, best vertex cover, and improved solutions, respectively, the space complexity is $O(|V|)$.

6 EMPIRICAL EVALUATION

6.1 Platform Description

All the algorithms in this project are run in macOS Catalina equipped with a CPU of 2.5 GHz Quad-Core Intel Core i7, a RAM of 16 GB 1600 MHz DDR3. The chosen programming language is Python 3.7.4 with the interpreter of Clang 4.0.1. The Python library **networkx** is utilized to store the graph structure.

6.2 Evaluation Criteria

6.2.1 Run time. The run time is recorded during the run of each algorithm. A cutoff time of 10 mins (600 seconds) is placed as a threshold. Any run beyond this criteria is terminated with the current solution returned. Moreover, several time logs are the recorded together with the corresponding solutions during the run once an improved solution appears. The time-lapsed results are still in progress and will not be shown in this partial report.

6.2.2 Relative Error. The relative error (RelErr) is imported to evaluate the solution quality of each algorithm in approximating the optimal solution. In this partial project report, we present the best-quality solution for each algorithm within the cutoff time (10 mins). The relative error is calculated as **(SOL - OPT) / OPT**.

Algorithm 5: Simulated annealing

```
Data: graph G, cutoff time, random seed
Result: minimum vertex cover
vertexCover <- V;
improvedSolution <- [];
improvedSolution add (curTime, vertexCover size);
bestVC <- vertexCover;
edgeNum <- |E|;
temp <- 100;
endTemp <- 1e-3;
decreasingRatio <- 0.99999;
initialize random with random seed;
while temp < endTemp AND runTime < cutoff time do
    temp <- temp * decreasingRatio;
    randomVertex <- randomly selected vertex in V;
    if randomVertex in vertexCover then
        remove randomVertex from vertexCover;
        if vertexCover not valid then
            add randomVertex back to vertexCover;
        end
    end
    else
        vertexCover add randomVertex;
        p <- math.exp(-(1 - randomVertex degree / edgeNum)
          / temp);
        if p < a random value from {0, 1} then
            remove randomVertex from vertexCover;
        end
    end
    if vertexCover size < bestVC size then
        bestVC = vertexCover;
        improvedSolution add (curTime, bestVC size);
    end
end
return bestVC size, bestVC, improvedSolution
```

6.3 Obtained Results

For heuristic approximation (Approx), hill climbing (LS1), simulated annealing (SA), ten random seeds (1, 2, 4, 8, 16, 32, 64, 128, 256, 512) are utilized (accordingly ten runs for each graph instance) to observe the effect of randomness on the quality of solutions. Since branch and bound (BnB) is a deterministic algorithm, only one run for each graph instance is conducted.

6.3.1 Lower bound from Heuristic Approximation. The best-quality comprehensive results and associated lower bounds of heuristics with approximation guarantees on MVC problem within the cutoff threshold (10 mins) are presented in Table 3 and Table 4, respectively.

6.3.2 Results of Branch and Bound. The comprehensive results of algorithm branch and bound (BnB) on MVC problem within the cutoff threshold (10 mins) are presented in Table 5.

Table 3. The 10-run average and best-quality (in bracket) run time, VC value, and relative error (RelErr) derived from heuristics with approximation guarantees for each benchmark graph instance.

Heuristics with Approximation			
Dataset	Time (s)	VC Value	RelErr
jazz	0.016 (0.016)	180 (176)	0.14 (0.11)
karate	1.63e-4 (1.38e-4)	21 (20)	0.51 (0.43)
football	0.003 (0.003)	108 (106)	0.15 (0.15)
as-22july06	4.09 (4.05)	6039 (5980)	0.83 (0.81)
hep-th	3.05 (3.07)	5771 (5724)	0.47 (0.46)
star	13.56 (13.95)	10224 (10196)	0.48 (0.48)
star2	9.28 (9.79)	6861 (6822)	0.51 (0.50)
netscience	0.24 (0.24)	1203 (1192)	0.34 (0.33)
email	0.15 (0.15)	829 (816)	0.39 (0.37)
delaunay n10	0.12 (0.12)	921 (910)	0.31 (0.29)
power	1.02 (1.01)	3621 (3598)	0.64 (0.63)

Table 4. The 10-run average and best-quality lower bound derived from heuristics with approximation guarantees for each benchmark graph instance.

Heuristics with Approximation		
Dataset	10-run avg. lower bound	best lower bound
jazz	90	88
karate	11	10
football	54	53
as-22july06	3019	2990
hep-th	2886	2862
star	5112	5098
star2	3431	3411
netscience	601	596
email	414	408
delaunay n10	461	455
power	1811	1799

Table 5. The run time, VC value, and relative error of branch and bound for each benchmark graph instance.

Branch and Bound			
Dataset	Time (s)	VC Value	RelErr
jazz	600	167	0.057
karate	0.099	14	0
football	600	98	0.043
as-22july06	600	5688	0.72
hep-th	600	5606	0.43
star	600	10748	0.56
star2	600	6806	0.50
netscience	600	1214	0.35
email	600	816	0.37
delaunay n10	600	907	0.29
power	600	3692	0.68

Table 6. The 10-run average and best-quality (in bracket) run time, VC value, and relative error of hill climbing for each benchmark graph instance.

Hill Climbing			
Dataset	Time (s)	VC Value	RelErr
jazz	0.002 (0.002)	160 (160)	0.013 (0.013)
karate	1.86e-4 (1.66e-4)	14 (14)	0 (0)
football	8.65e-4 (9.44e-4)	96 (95)	0.021 (0.011)
as-22july06	5.58 (5.53)	3331 (3327)	0.0085 (0.0073)
hep-th	1.07 (1.06)	3941 (3939)	0.0038 (0.0033)
star	4.46 (4.20)	7232 (7218)	0.048 (0.046)
star2	4.88 (5.33)	4856 (4845)	0.069 (0.067)
netscience	0.041 (0.039)	899 (899)	0 (0)
email	0.026 (0.024)	616 (613)	0.037 (0.032)
delaunay n10	0.027 (0.025)	744 (741)	0.058 (0.054)
power	0.38 (0.38)	2280 (2267)	0.035 (0.029)

Table 7. The 10-run average and best-quality (in bracket) run time, VC value, and relative error of simulated annealing for each benchmark graph instance.

Simulated Annealing			
Dataset	Time (s)	VC Value	RelErr
Jazz	8.65 (8.15)	158 (158)	0 (0)
Karate	1.09 (0.093)	14 (14)	0 (0)
Football	4.18 (3.41)	94 (94)	0 (0)
as-22july06	387.68 (356.61)	3454 (3439)	0.046 (0.041)
hep-th	172.66 (165.60)	4034 (4018)	0.028 (0.023)
star	369.04 (372.22)	7473 (7463)	0.083 (0.081)
star2	357.40 (354.64)	4775 (4766)	0.051 (0.049)
netscience	28.11 (31.55)	903 (901)	0.0044 (0.0022)
email	26.73 (26.28)	605 (601)	0.019 (0.012)
delaunay n10	23.67 (23.20)	733 (727)	0.043 (0.034)
power	83.38 (81.47)	2310 (2301)	0.049 (0.044)

6.3.3 Results of Hill Climbing. The 10-run average and best-quality (in bracket) run time, VC value, and relative error (RelErr) of algorithm hill climbing (LS1, HC) on MVC problem within the cutoff threshold (10 mins) are presented in Table 6.

6.3.4 Results of Simulated Annealing. The10-run average and best-quality (in bracket) run time, VC value, and relative error (RelErr) of algorithm simulated annealing (LS2, SA) on MVC problem within the cutoff threshold (10 mins) are presented in Table 7.

7 DISCUSSION

7.1 Branch and Bound

The Table 5 shows that the branch and bound algorithm performed the slowest among the four algorithm. Given a cutoff threshold 600 seconds, the processing time remarkably expands for large graphs. The overall RelErr are relatively high comparing to other algorithms, among which the highest RelErr can reach 0.7221 (as-22july06), which is ascribed to incomplete run of instances withe

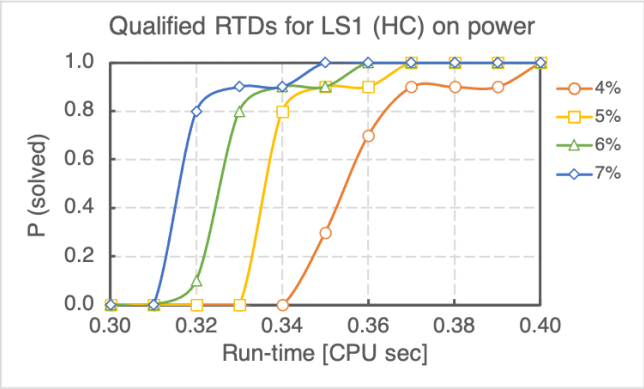

Fig. 2. Qualified run-time distributions (QRTDs) for LS1 (hill climbing) on power.graph.

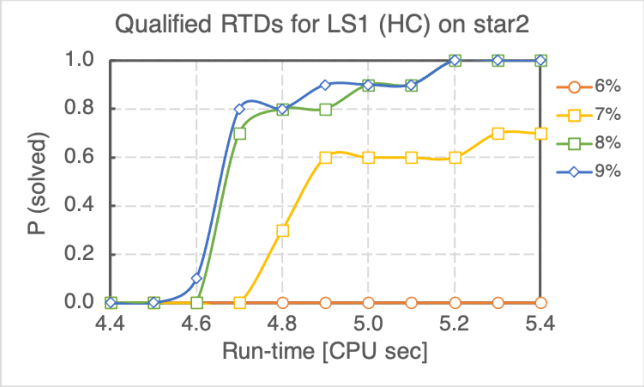

Fig. 3. Qualified run-time distributions (QRTDs) for LS1 (hill climbing) on star2.graph.

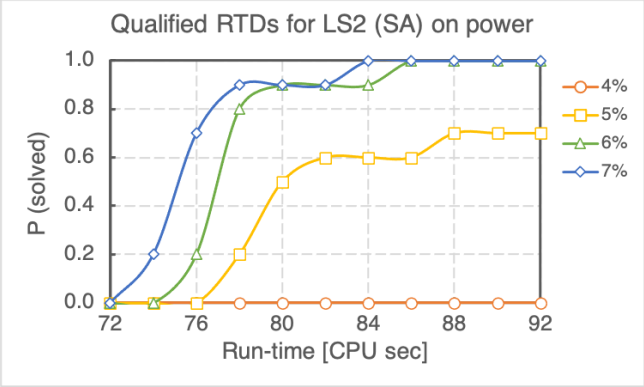

Fig. 4. Qualified run-time distributions (QRTDs) for LS2 (simulated annealing) on power.graph.

the BnB algorithm. It's reasonable that branch and bound algorithm sacrifices the running time to guarantee a valid optimal solution. In our implementation, the branch and bound algorithm shows little

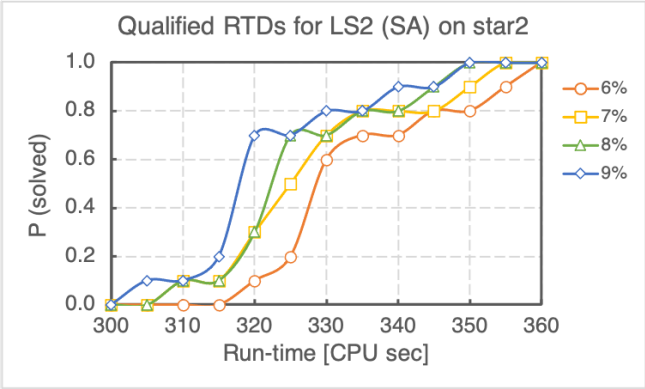

Fig. 5. Qualified run-time distributions (QRTDs) for LS2 (simulated annealing) on star2.graph.

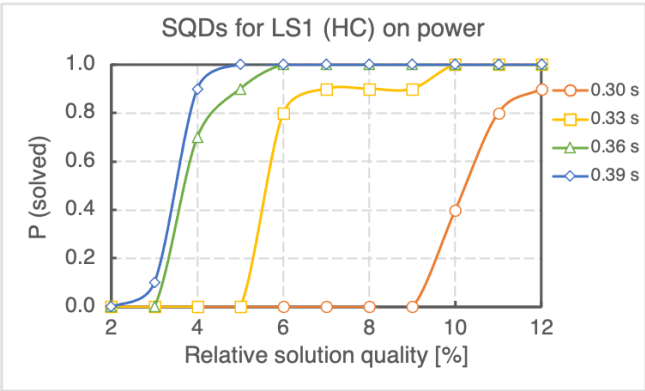

Fig. 6. Solution quality distributions (SQDs) for LS1 (hill climbing) on power.graph.

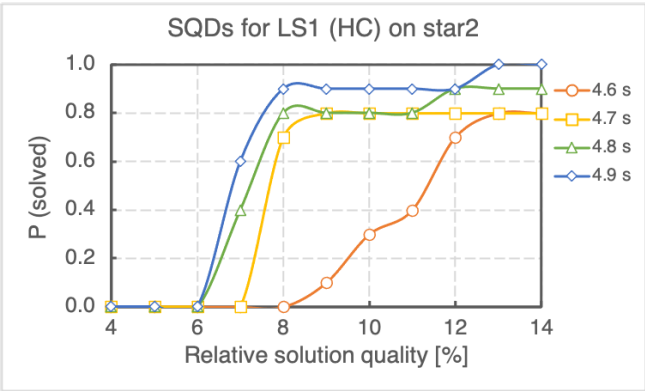

Fig. 7. Solution quality distributions (SQDs) for LS1 (hill climbing) on star2.graph.

strength over the approximation algorithm within the 600-second cutoff time for coping with large graphs since it runs in exponential time. However, it is reasonable to speculate that our BnB algorithm

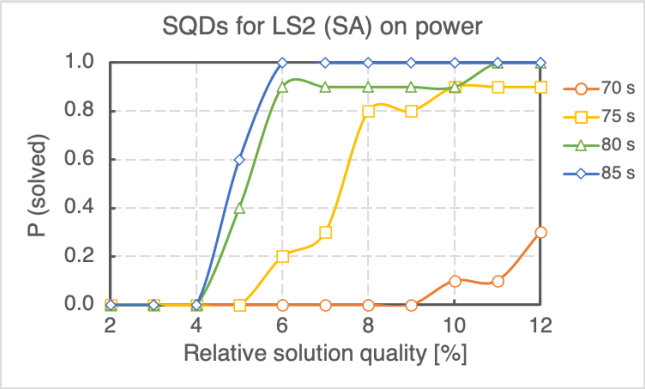

Fig. 8. Solution quality distributions (SQDs) for LS2 (simulated annealing) on power.graph.

Fig. 9. Solution quality distributions (SQDs) for LS2 (simulated annealing) on star2.graph.

would generate optimal solutions for each graph instance if given enough time to finish every run.

7.2 Heuristics with Approximation Guarantees

The Table 3 shows that the run time of heuristic approximation grows considerably with the expanding size of the graph. However, even for the large graph *star*, the run time (14 s) is still below the set cutoff threshold (10 mins), which is consistent with its theoretical polynomial time complexity ($O(|E|^2)$). Compared to the Table 1, our heuristic approximation algorithm generates a relatively tighter lower bound for the optimal solutions (4).

Our implemented heuristic approximation incorporates randomness in selecting an edge from the uncovered edge set in every iteration. Both the Table 3 and (4) show that different selection of random seed does influence the running time and the VC values. In the Table 4, we showed two different lower bound. The 10-run avg. lower bound is half of each 10-run average results and the best lower bound is half of each best result. However, the difference between the 10-run average results and best results are not significant.

In general, in coping with this minimum vertex cover problem, the heuristic approximation does show its advantage of providing a

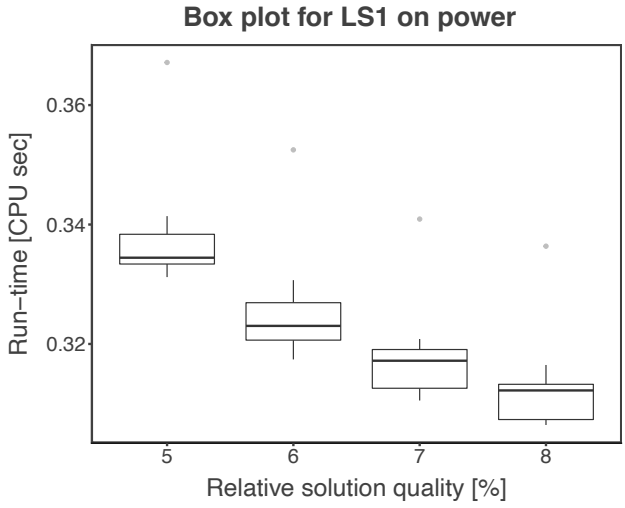

Fig. 10. Box plot for LS1 (hill climbing) on power.graph.

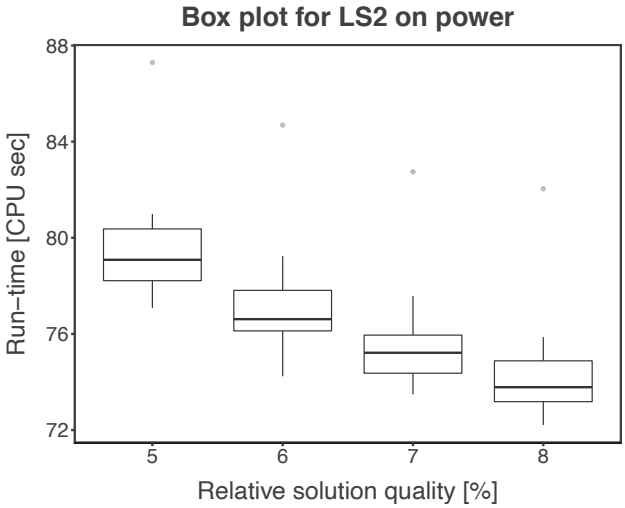

Fig. 12. Box plot for LS2 (simulated annealing) on power.graph.

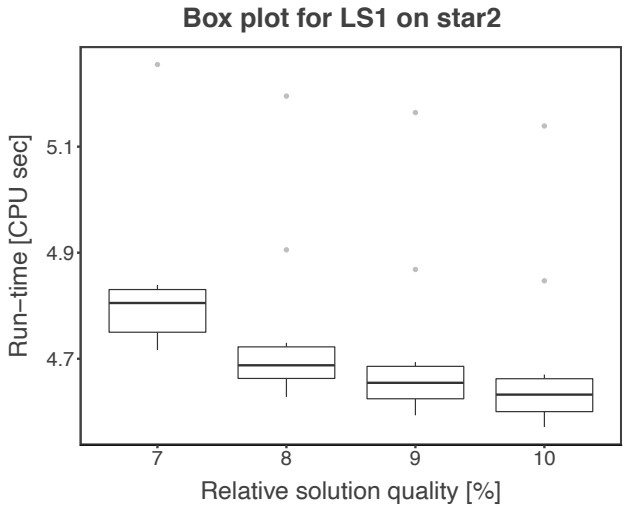

Fig. 11. Box plot for LS1 (hill climbing) on star2.graph.

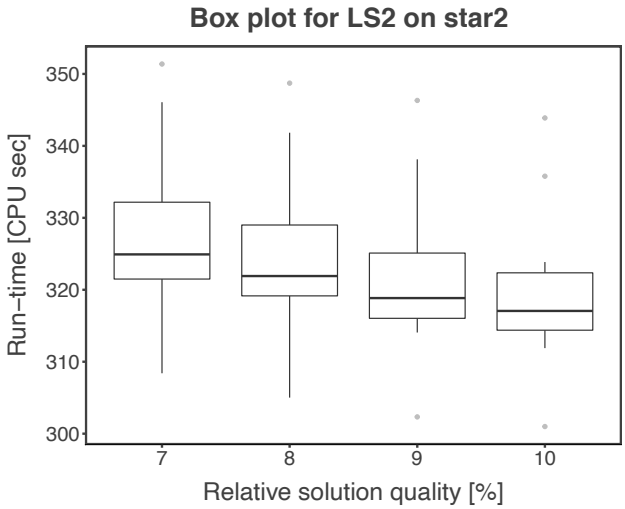

Fig. 13. Box plot for LS2 (simulated annealing) on star2.graph.

bound on the quality of the solution within a polynomial running time. Since,for each graph, we provided two lower bound, the 10-run avg.lower bound and the best lower bound and the difference between those two different lower bounds are not significant, we can choose either one as the lower bound. It would be helpful to accelerate the search of optimal solution with the branch and bound algorithm.

7.3 Hill Climbing

The Table 6 shows that the hill climbing algorithm is able to approximate the near-optimal solution for all benchmark instances with a relatively high accuracy. The maximum RelErr among the 11 graph instances is < 0.07. In addition, the hill climbing algorithm is time efficient in tackling the MVC problem (consistent with the theoretical polynomial time complexity of $O(|V|*(max(|V|+|E|)))$. Even for the larget graph instances like "as-22july06" and "star2", it only takes HC < 6 s to achieve the best-quality results.

As shown in the Table 6, different random seed (resulting in different selection of least degree vertex to be potentially removed) does lead to different VC values and the associated run time. However, both the run time and the VC value for the 10-run average results and the best results display no significant change, which indicates that the initialization of random seed does not overall affect the solution quality of the hill climbing algorithm within the acceptable error.

From the figures of qualified runtime distributions (QRTDs) for LS1 (hill climbing) algorithm (Figure 2 for power.graph and Figure 3 for star2.graph), it can be concluded that the probability of achieving solutions all increases with the runtime for each specified relative solution quality. Besides, it can be observed that the lower the relative solution quality, takes the longer time to reach. The same observations also hold true for the figures of QRTDs for LS2 (simulated annealing) algorithm (Figure 4 for power.graph and Figure 5). Through the comparison of the Figure 2 and the Figure 4, it can be derived that the hill climbing (LS1) algorithm can reach a specified relative solution quality much quicker than the simulated annealing (LS2) algorithm in coping with the power.graph. Besides, the hill climbing algorithm can reach the relative solution quality of 4%, while the simulated annealing in unable to achieve that. This result might be explained by simulated annealing accepting a worse neighboring solution but not leading to a solution more superior than the local minimum the hill climbing algorithm achieved. On the contrary, the comparison between the Figure 3 and the Figure 5 show that although the hill climbing can solve the problem quicker, it is unable to achieve the relative solution quality of 6% for the star2.graph. However, the simulated annealing is successful to reach this quality. In this case, the step of accepting a worse neighboring solution ultimately helps to result in a better optimal compared to the local optimal the hill climbing achieved.

The figures of solution quality distributions (SQDs) for LS1 (hill climbing) (Figure 6 for power.graph and Figure 7 for star2.graph) suggest that with the loosing of the quality requirement (increasing of the relative solution quality), the probability of achieving solutions increases for each run time scenario. In addition, both the Figure 6 and the Figure 7 show that the longer run time has higher fraction of achieved solutions compared to the shorter run time given the same relative solution quality. These observations also hold true for the figures of SQDs for LS2 (simulated annealing) algorithm (Figure 8 for power.graph and Figure 9). Since there is almost two magnitude difference between the run time of hill climbing and simulated annealing, the SQDs for the two algorithms are drawn with different runtime ranges, and their results can not be compared.

The box plots of run time for LS1 (hill climbing) (Figure 10 for power.graph and Figure 11 for star2.graph) indicate that it takes shorter average time (mean of 10 runs with different random seeds) to achieve the solution with a lower quality requirement (higher value of the relative solution quality). Both the Figure 10 and the Figure 11 have outliers that takes more time to achieve a specified relative solution quality compared to the majority time distributions. It suggests that some random choice of least degree vertex "slows down" the process of achieving the final solution of local minimum. The decreasing trend of average run time with the increasing relative solution quality is also observed in the box plots of run time for LS2 (simulated annealing) (Figure 12 for power.graph and Figure 13 for star2.graph). The decreasing tendency is slightly weakened for simulated annealing to cope with star2.graph. It might be ascribed to larger variance of run time as indicated by the larger range (expand) of the run time. In Figure 12, one longer time outlier appears, while in Figure 13, both longer and shorter time outlier appear. It implies that the selection of random seed may either "accelerates" or "slows down" the process of achieving the final solution.

7.4 Simulated Annealing

The table 7 shows that the Simulated Annealing algorithm is a good solution with a high accuracy. The run time increases more obviously with the increasing size of the graph instance compared to hill climbing algorithm but is still better than the branch and bound due to the polynomial time complexity ($O(|T| * (max(|V| + |E|))))$. The largest RelErr from all examples is 0.0832, which is much less than Heuristics with Approximation, and Branch and Bound. However, since the reason we consider Simulated Annealing is that we assumed that this algorithm would performs better than Hill Climbing, the result shows that the time cost longer and the RedErr is not better than Hill Climbing. Therefore, Hill Climbing may be overall better than Simulated Annealing.

As shown in the Table 7, different random seed (resulting in different selection of least degree vertex to be potentially removed) does lead to different VC values and the associated run time. However, both the run time and the VC value for the 10-run average results and the best results show litte difference, which implies that the initialization of random seed does not overall affect the solution quality of the hill climbing algorithm within the acceptable error.

The explanations of QRTDs, SQDs, and boxplot for the LS2 (simulated annealing) and their comparisons to the LS1 (hill climbing) are discussed in 7.3, and are omitted here. In general, our implemented hill climbing algorithm is superior than the simulated annealing algorithm in both run time and solution quality (although the simulated annealing performs better in some graph instances). In this sense, it can be formulated that hill climbing is probabilistically dominates simulate annealing in our implementation. (RTD of hill climbing is "above" that of simulated annealing).

8 CONCLUSION

Our implemented heuristic approximation proves to be an efficient algorithm in generating a relatively tight lower bound for the optimal MVC solution with a good time efficiency (polynomial), which would be beneficial for the search of optimal solutions such as branch and bound algorithm.

Our branch and bound algorithm is able to generate the optimal MVC solutions for the small graphs without exceeding the cutoff time threshold. The larger RelErr for larger graph instances is ascribed to incomplete run of the algorithm. Given enough time, our BnB algorithm is supposed to achieve the optimal solution for each graph instance. The time insufficiency nature of BnB is consistent with the theoretical exponential time complexity (2^n) in worse scenario. We may try other strategies to "choose" and "expand" the subproblems to optimize our algorithm.

Both hill climbing and simulated annealing are proved to be two good local search algorithms in searching for near-optimal solution for the minimum vertex cover problem due to their high accuracy and time efficiency. Our implemented hill climbing performs better than the simulated appealing, although in theory the simulated appealing is supposed to have a better solution quality (help to escape the local minimum). Selection of random seeds does result

in difference in the run time and the solution quality, however, the difference is relatively insignificant within the error.

REFERENCES

Michael R Garey and David S Johnson. 1979. *Computers and intractability*. Vol. 174. freeman San Francisco.

George Karakostas. 2005. A better approximation ratio for the vertex cover problem. In *International Colloquium on Automata, Languages, and Programming*. Springer, 1043–1050.

Marek Karpinski and Alexander Zelikovsky. 1996. *Approximating dense cases of covering problems*. Inst. für Informatik.

Christos H Papadimitriou and Kenneth Steiglitz. 1998. *Combinatorial optimization: algorithms and complexity*. Courier Corporation.

Xinshun Xu and Jun Ma. 2006. An efficient simulated annealing algorithm for the minimum vertex cover problem. *Neurocomputing* 69, 7-9 (2006), 913–916.