CSE6140 Final Project

AVERY BODENSTEIN*, Georgia Institute of Technology, USA
ADRIAN THINNYUN*, Georgia Institute of Technology, USA
JAI JACOB*, Georgia Institute of Technology, USA
ZHEYI ZHANG*, Georgia Institute of Technology, USA

A clear and well-documented LaTeX document is presented as an article formatted for publication by ACM in a conference proceedings or journal publication. Based on the "acmart" document class, this article presents and explains many of the common variations, as well as many of the formatting elements an author may use in the preparation of the documentation of their work.

CCS Concepts: • **Theory of computation → Graph algorithms analysis**; **Branch-and-bound**; **Approximation algorithms analysis**.

Additional Key Words and Phrases: minimum vertex cover, spanning, topology, heuristics

ACM Reference Format:
Avery Bodenstein, Adrian Thinnyun, Jai Jacob, and Zheyi Zhang. 2022. CSE6140 Final Project. *Proc. ACM Meas. Anal. Comput. Syst.* 37, 4, Article 111 (August 2022), 21 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

The Minimum Vertex Cover (MVC) problem is a classical NP-complete problem with a wide variety of real-world and theoretical applications. The problem consists of finding the smallest subset of vertices in a graph that includes at least one endpoint of every edge in the graph. It is the optimization version of the Vertex Cover problem, which instead asks if there exists a subset of vertices of size less than or equal to a given positive integer that includes at least one endpoint of every edge in the graph.

The problem is in NP, meaning that a solution to the problem cannot be found with an algorithm that runs in polynomial time if P ≠ NP. Moreover, the problem is in NP-hard, meaning that every problem in NP is polynomially reducible to MVC (i.e. any instance of an NP problem can be converted into an instance of MVC in polynomial time). Since MVC is both in NP and NP-hard, it is also NP-complete, and was included as one of Karp's original 21 NP-complete problems [10].

In this project, we implemented and tested four different approaches to the Minimum Vertex Cover problem. The first approach is a Branch and Bound (BnB) algorithm which runs in exponential time in the worst-case but may perform significantly better than a brute force search and is guaranteed to provide an optimal solution. The second approach is a constructive heuristic algorithm, in particular the *Greedy Independent Cover (GIC)* algorithm

*All authors contributed equally to this research.

Authors' addresses: Avery Bodenstein, abodenstein3@gatech.edu, Georgia Institute of Technology, North Ave NW, Atlanta, Georgia, USA, 30332; Adrian Thinnyun, Georgia Institute of Technology, North Ave NW, Atlanta, Georgia, USA, 30332; Jai Jacob, Georgia Institute of Technology, North Ave NW, Atlanta, Georgia, USA, 30332; Zheyi Zhang, Georgia Institute of Technology, North Ave NW, Atlanta, Georgia, USA, 30332.

2476-1249/2022/8-ART111 $15.00
https://doi.org/XXXXXXX.XXXXXXX

proposed by Halldórsson and Radhakrishnan [6], which runs in polynomial time and produces a solution with an approximation guarantee. The final two approaches are local search algorithms, one involving Simulated Annealing and the other involving Genetic Algorithms.

[Insert paragraph about results]

2 PROBLEM DEFINITION

Given a graph $G = (V, E)$, a vertex cover of G is a subset of vertices $A \subseteq V$ such that for every edge $(u, v) \in E$, $u \in A \vee v \in A$. The Minimum Vertex Cover problem is then the problem of finding a vertex cover S of G such that for every vertex cover A of G, $|S| \leq |A|$, i.e. S is the smallest vertex cover of G. In other words, the problem consists of finding a vertex cover of G using as few vertices as possible.

3 RELATED WORK

The Vertex Cover problem was introduced in Karp's 21 NP-complete problems as the "Node Cover" problem [10]. The problem was defined as follows:

```
NODE COVER
INPUT: graph G', positive integer l
PROPERTY: There is a set R ⊆ N' such that |R| ⊆ l and
every arc is incident with some node in R.
```

As part of Karp's result that the Satisfiability problem is reducible to each of the other problems listed in the paper, he showed that the Clique problem is reducible to the Node Cover problem, and that the Node Cover problem is reducible to the Feedback Node Set problem, the Feedback Arc Set problem, the Directed Hamilton Circuit problem, and the Set Covering problem.

Since its introduction, the Minimum Vertex Cover problem has had a wide range of theoretical and practical results related to it. In 1983, Bar-Yehuda and Even [2] presented an algorithm that produces a solution to the problem with an approximation ratio of $2-\Theta(\frac{\log\log n}{\log n})$. A similar result was achieved by Monien and Speckenmeyer [11]. With Δ as the maximal degree of the graph, Hochbaum [8] was able to approximate the problem with a ratio of $(2 - \frac{2}{\Delta})$. This result was improved by Halldórsson and Radhakrishnan [6] whose algorithm achieves an approximation ratio of $2 - \frac{\log\Delta+O(1)}{\Delta}$, and then again by Halperin [7] who achieved an approximation ratio of $(2 - (1 - o(1))\frac{2\ln\ln\Delta}{\ln\Delta})$ by using semidefinite programming relaxations. This result was improved further by Karakostas [9] who used an even stronger semidefinite programming relaxation to achieve an approximation ratio of $2 - \Theta(\frac{1}{\sqrt{\log n}})$.

Several local search algorithms have been proposed for the Minimum Vertex Cover problem. One such algorithm is Edge Weighting Local Search (EWLS) [3], which is an iterated search algorithm based on the idea of extending a partial vertex cover into a full vertex cover. Another algorithm proposed for MVC is NuMVC [4], which uses a two-stage exchange strategy and edge weighting with forgetting to improve upon state-of-the-art algorithms. A stochastic local search algorithm named COVER (Cover Edges Randomly) [12], which combines several heuristic criteria and a healthy dose of randomness to strike a balance between guided search and diversity.

4 ALGORITHMS

Detailed description of each algorithm you have implemented, with pseudo-code, approx- imation guarantee (if any), time and space complexities, etc. What are the potential strengths and weaknesses of each type of approach? Did you use any kind of automated tuning or configuration for your local search? Why and how you chose your

local search approaches and their components? Please cite any sources of information that you used to inform your algorithm design

4.1 Branch and Bound

4.1.1 Description. The branch and bound approach is a non-polynomial time algorithm which is guaranteed to return an optimal solution. The BnB algorithm at worst does not perform any better than a brute force search, but can have significant performance gains where portions of the search space can be pruned. This pruning is performed by iteratively defining sub-problems and determining lower bounds on the cost to go for these sub-problems. If the best possible cost for a sub-problem exceeds the current best cost that subproblem (and therefore all it's derivatives) can be pruned from the search space. The sub-problems presented in this algorithm are defined based on inclusion or exclusion of nodes from the cover set.

4.1.2 *Pseudo Code.* -

The overall branch and bound algorithm has the following structure:

```
Data: P
F ← (∅, P);
B ← (+∞, (∅, P));
while F not empty do
    choose (X, Y) in F;
    expand (X, Y);
    let (x_1, y_1), (x_2, y_2) be new configurations;
    foreach (x_i, y_i) do
        if solution found then
            if cost(x_i) < B then
                B ← (cost(x_i), (x_i, y_i));
            end
        end
        if not dead end then
            if lowerBound(x_i) < B then
                F ← F ∪ (x_i, y_i);
            end
        end
    end
end
return (B)
```

Algorithm 1: Branch and Bound

where P is the graph, x_i are all the vertices in the cover set, and y_i are the vertices remaining for selection. There are four sub functions, "choose", "expand", "checkSolution", "checkDeadEnd", and "lowerBound". In the implementation presented here, choose selects the subproblem in F with the lowest lowerBound on cost to go. Expand takes that subproblem and returns two subproblems, one with the node with fewest unique edges selected and one without that node as a possible selection. checkSolution checks to see if all edges are covered by x_i. checkDeadEnd checks if $x_i \cup y_i$ covers all edges. Finally lowerBound runs the heuristic algorithm described in this report on y_i then divides by the approximation ratio (2) to get a lower bound on the possible additional nodes required to cover all edges. This is then added to the number of nodes already in x_i.

4.1.3 *Algorithm Analysis.* Additional description

Gives exact solution

Time and space complexity

4.2 Construction Heuristic

4.2.1 Description. The construction heuristic approach implemented here is the *Greedy Independent Cover (GIC)* algorithm, presented in Halldórsson and Radhakrishnan [1994] [6] for the independent set problem. This is an

approximation approach which runs in polynomial time and yields an answer within a bound of the optimal solution. Runtime and Approximation Ratio analysis is provided in 4.2.3. This algorithm lends itself well to use with priority queues. All vertices in the graph are stored in a priority queue, ordered by the number of adjacent edges remaining in G: q[nodeLabel] = [nEdges,destinationNodes]. At each iteration the vertex with the fewest remaining edges is popped from the queue. Each of it's neighbors is then removed from the queue and added to the cover set. For each remaining edge in each neighbor, the source node is removed from the list of edges in the destination node and the number of edges in the destination node is lowered by one. This process continues until all edges are covered.

4.2.2 *Pseudo Code.* -

```
Data: G = (V,E)
C ← ∅;
while E ≠ ∅ do
    select a vertex u of minimum degree;
    C ← C ∪ N(u);
    V ← V - (N(u) ∪ u);
end
return C;
```

Algorithm 2: Greedy Independent Cover (GIC)

Where N(u) is the neighborhood (all adjacent vertices to) u.

```
Data: q,nEdges
C ← 0;
while C < nEdges do
    u ← q.peek();
    foreach neigbor i in N(u) do
        C ← C + q[i][0];
        foreach edge (i,j) in neighbor do
            q[j][0] ← q[j][0] - 1;
            q[j][1] ← q[j][1] - {i};
        end
    end
    q = q - u;
end
return C;
```

Algorithm 3: Detailed Implementation

4.2.3 *Algorithm Analysis.* -

As discussed in Delbot and Laforest [2010] [5] the Greedy Independent Cover algorithm performs extremely well. As can be seen in table 1 the relative error for GIC never exceeds 6% (and this is on the Jazz graph where the estimated solution is only 1 node larger than the optimal solution). However, the disadvantage of GIC is that the worst case performance is relatively poor. As shown in Avis and Imamura [2007] [1], the approximation ratio is at least $\frac{\sqrt{\Delta}}{2}$ where Δ is the maximum degree in G.

Figure 1 shows a comparison between the approximation ratios of several common algorithms described in [5]. Note that while GIC performs quite well, it's approximation ratio exceeds Depth First Search, or either of the Edge Deletion bounds for any Δ above 16.

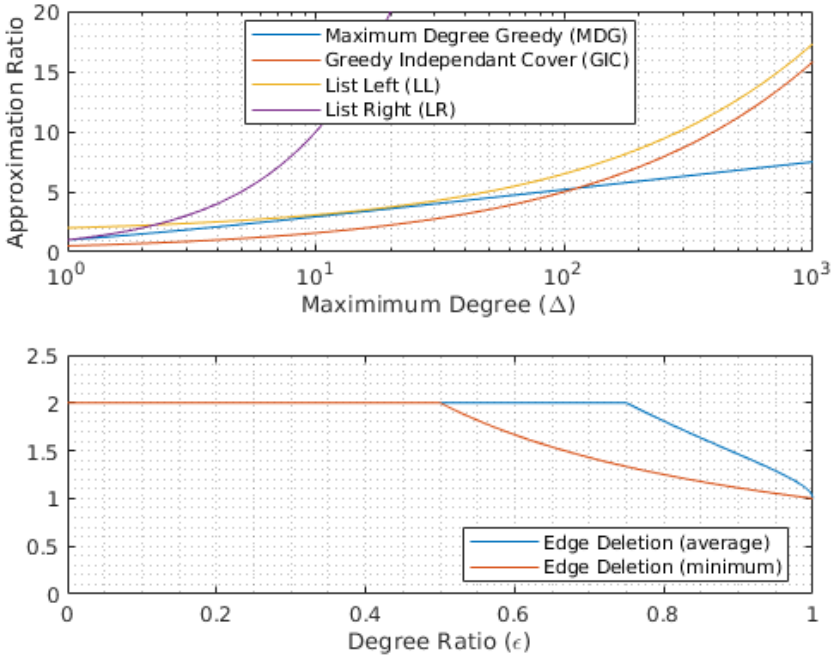

Fig. 1. Comparison of approximation ratios between several Vertex Cover heuristics

The time complexity of the GIC algorithm as implemented is $O(n^2)$ where n is the number of nodes. As each node is selected via the minimum cardinality, at worst it has zero neighbors, so there would be at worst n node selections. However in this case there would be 0 edge updates. For every neighbor that is present there are edge updates equal to the neighbors' cardinality. This can be at most $n - i$, where i is the number of previous node selections. The complete complexity is then $O(\frac{n}{2}(n+1))$.

4.3 Local Search 1 (Simulated Annealing)

4.3.1 Description. The first local search algorithm we implement is simulated annealing. The main task is to find an appropriate cost evaluation function, and a definition of neighbor.

For the first question, we define the cost as

$$|V'| + \alpha \texttt{ number of edges uncovered}$$

In the equation, α is a factor to define how important it is to cover all the edges while exploring new solutions. Then the main procedure is to start with an initial solution with a greedy algorithm and a large temperature, lower it gradually in each loop, find a neighbor and evaluate it, choose to switch to it or not using the probability given the evaluated cost change.

For the second question, we define neighbor in this algorithm as a set of vertices that differ from the current set with exactly one vertex, that is, the neighbor either has one more vertex in the set, or has all but one vertex compared with the current one. Using this definition, we can possibly go to the optimal solution in the sense that it is connected to the initial solution via the neighbor relationship.

Data: V,E
init_sol = init(V, E, neighbors);
best = (len(*init_sol*), *init_sol*);
v_set = *init_sol*;
edges = edges covered by *v_set*;
T = T_{max};
while $T > 0$ **do**
  next = random_of($\{1, 2, \ldots, |V|\}$);
  gain = 0;
  if *next* $\in$ *v_set* **then**
    gain+ = 1;
    for $j \in neighbors[next]$ **do**
      if $j \notin v\_set$ **then**
        gain− = α
      end
    end
  end
  else
    gain− = 1;
    for $j \in neighbors[next]$ **do**
      if $j \notin v\_set$ **then**
        gain+ = α
      end
    end
  end
  $p = \min\{1, e^{\frac{gain}{T}}\}$;
  if *random(0,1)* < *p* **then**
    if *next* $\in$ *v_set* **then**
      v_set.remove(*next*);
      for $j \in neighbors[next]$ **do**
        if $j \notin v\_set$ **then**
          edges.remove(*next*, *j*)
        end
        ;
      end
    end
    else
      v_set.add(*next*);
      for $j \in neighbors[next]$ **do**
        if $j \notin v\_set$ **then**
          edges.add(*next*, *j*)
        end
      end
    end
  end
  if *len(edges)* == $|E|$ ***and*** *len(v_set)*<*best*[0] **then**
    best = $(len(v\_set), v\_set)$
  end
  $T- = T_{MAX}/k$;
end
return (*best*)

Algorithm 4: Simulated Annealing

4.3.2 Pseudo Code.

4.3.3 Algorithm Analysis. The pseudocode in last section shows how one whole loop of simulated annealing works, after each loop, if there's still more time available, the program simply start a new loop to try finding a better solution. Now let's analyze its time complexity of one complete loop. For a specific temperature, the main cost is for a randomly chosen vertex $next$, visiting all of its neighbors, which takes $O(|neighbors[next]|)$ time; then if the gain is positive or neighbor gets lucky, we update current vertex set and edge set, for which the main cost is also to visit all neighbors of $next$. Since this takes k iterations, if we assume all vertices are visited almost evenly in long term, then on average the time complexity is $O(k|E|/|V| + |V| + |E|)$, where $|V| + |E|$ comes from the greedy algorithm to get the initial solution. As for space complexity, the main cost is to store the whole graph and the current vertex and edge set, which in total takes $O(|V| + |E|)$ space.

Overall, the strength of this local search algorithm is time efficiency, since each complete loop only takes $O(k|E|/|V| + |V| + |E|)$ time; moreover, as said before, because it is possible to visit any neighbor, and the initial solution is connected to an optimal solution under the neighbor relationship, it is guaranteed possible to reach the optimal solution. On the other hand, because this algorithm takes many hyper parameters T_{MAX}, k, α, it is hard to find a good setting of them, especially when it encounters another input graph. Another weakness is the low speed of convergence to the optimal solution– because of how neighbor is defined, each iteration updates at most one vertex, which makes it hard to converge to the potentially very different optimal solution.

4.4 Local Search 2 (specific descriptor)

4.4.1 Description. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

4.4.2 Pseudo Code.

```
   Data: B,C
1  n_a = 0;
2  i = len(B);
3  j = len(C);
4  A = [0]*(i+j-2);
5  for l = i+j-2; l ≥ 0; l-=1 do
6      if i == 0 then
7          A[0:l] = C[0:j];
8          return (A,n_a)
9      else if j == 0 then
10         A[0:l] = B[0:i];
11         return (A,n_a)
12     if C[j-1] ≥ B[i-1] then
13         A[l] = C[j-1];
14         j = j-1;
15     else
16         A[l] = B[i-1];
17         n_a = n_a + j;
18         i = i-1;
19 end
20 return (A,n_a)
```

Algorithm 5: merge

4.4.3 Algorithm Analysis. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

5 EMPIRICAL EVALUATION

a detailed description of your platform (CPU, RAM, language, compiler, etc.), experimental procedure, evaluation criteria and obtained results (plots, tables, etc.). What is the lower bound on the optimal solution quality that you can drive from the results of your approximation algorithm and how far is it from the true optimum? How about from your branch-and-bound?

Table 1. Algorithm Performance

	Branch and Bound			Construction Heuristic			Local Search 1(avg of 10 runs)			Local Search	
Dataset	Time(s)	VC Value	RelErr	Time(s)	VC Value	RelErr	Time(s)	VC Value	RelErr	Time(s)	VC Value
jazz				0.0037	159	0.063	2.82	158.1	0.00063		
karate				0.0005	14	0.00	0.0.015	14.0	0.00		
football				0.0014	95	0.011	0.48	94.0	0.00		
as-22july06				0.21	3303	0.00	7.12	16729.1	4.06		
hep-th				0.098	3943	0.0043	4.98	5665.9	0.44		
star				0.24	7069	0.024	6.27	10440.8	0.51		
star2				0.21	4674	0.029	5.14	12055.4	1.65		
netscience				0.017	901	0.0022	4.68	986.3	0.097		
email				0.015	604	0.017	4.04	687.3	0.15		
delaunay n10				0.012	733	0.043	5.17	768.3	0.092		
power				0.051	2226	0.010	5.91	3108.2	0.41		

6 DISCUSSION

a comparative analysis of how different algorithms perform with respect to your evaluation criteria, or expected time complexity, etc

7 CONCLUSION

8 MATH EQUATIONS

You may want to display math equations in three distinct styles: inline, numbered or non-numbered display. Each of the three are discussed in the next sections.

8.1 Inline (In-text) Equations

A formula that appears in the running text is called an inline or in-text formula. It is produced by the **math** environment, which can be invoked with the usual `\begin . . . \end` construction or with the short form `$ . . . $`. You can use any of the symbols and structures, from α to ω, available in LaTeX [**?**]; this section will simply show a few examples of in-text equations in context. Notice how this equation: $\lim_{n\to\infty} x = 0$, set here in in-line math style, looks slightly different when set in display style. (See next section).

8.2 Display Equations

A numbered display equation—one set off by vertical space from the text and centered horizontally—is produced by the **equation** environment. An unnumbered display equation is produced by the **displaymath** environment.

Again, in either environment, you can use any of the symbols and structures available in LaTeX; this section will just give a couple of examples of display equations in context. First, consider the equation, shown as an inline equation above:

$$\lim_{n\to\infty} x = 0 \tag{1}$$

Notice how it is formatted somewhat differently in the **displaymath** environment. Now, we'll enter an unnumbered equation:

$$\sum_{i=0}^{\infty} x + 1$$

and follow it with another numbered equation:

$$\sum_{i=0}^{\infty} x_i = \int_0^{\pi+2} f \tag{2}$$

just to demonstrate LaTeX's able handling of numbering.

9 FIGURES

The "`figure`" environment should be used for figures. One or more images can be placed within a figure. If your figure contains third-party material, you must clearly identify it as such, as shown in the example below.

Your figures should contain a caption which describes the figure to the reader.

Figure captions are placed *below* the figure.

Every figure should also have a figure description unless it is purely decorative. These descriptions convey what's in the image to someone who cannot see it. They are also used by search engine crawlers for indexing images, and when images cannot be loaded.

A figure description must be unformatted plain text less than 2000 characters long (including spaces). **Figure descriptions should not repeat the figure caption – their purpose is to capture important information that is not already provided in the caption or the main text of the paper.** For figures that convey important and complex new information, a short text description may not be adequate. More complex alternative descriptions can be placed in an appendix and referenced in a short figure description. For example, provide a data table capturing the information in a bar chart, or a structured list representing a graph. For additional information regarding how best to write figure descriptions and why doing this is so important, please see https://www.acm.org/publications/taps/describing-figures/.

10 CITATIONS AND BIBLIOGRAPHIES

The use of BibTeX for the preparation and formatting of one's references is strongly recommended. Authors' names should be complete — use full first names ("Donald E. Knuth") not initials ("D. E. Knuth") — and the salient identifying features of a reference should be included: title, year, volume, number, pages, article DOI, etc.

The bibliography is included in your source document with these two commands, placed just before the `\end{document}` command:

```
\bibliographystyle{ACM-Reference-Format}
\bibliography{bibfile}
```

where "`bibfile`" is the name, without the "`.bib`" suffix, of the BibTeX file.

Citations and references are numbered by default. A small number of ACM publications have citations and references formatted in the "author year" style; for these exceptions, please include this command in the **preamble** (before the command "`\begin{document}`") of your LaTeX source:

```
\citestyle{acmauthoryear}
```

Some examples. A paginated journal article [5] [1] [6]

11 ACKNOWLEDGMENTS

Identification of funding sources and other support, and thanks to individuals and groups that assisted in the research and the preparation of the work should be included in an acknowledgment section, which is placed just before the reference section in your document.

This section has a special environment:

```
\begin{acks}
...
\end{acks}
```

so that the information contained therein can be more easily collected during the article metadata extraction phase, and to ensure consistency in the spelling of the section heading.

Authors should not prepare this section as a numbered or unnumbered `\section`; please use the "`acks`" environment.

ACKNOWLEDGMENTS

To Robert, for the bagels and explaining CMYK and color spaces.

REFERENCES

[1] David Avis and Tomokazu Imamura. 2007. A list heuristic for vertex cover. *Operations Research Letters* 35, 2 (2007), 201–204. https://doi.org/10.1016/j.orl.2006.03.014

[2] Reuven Bar-Yehuda and Shimon Even. 1983. *A local-ratio theorm for approximating the weighted vertex cover problem.* Technical Report. Computer Science Department, Technion.

[3] Shaowei Cai, Kaile Su, and Qingliang Chen. 2010. EWLS: A new local search for minimum vertex cover. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*.

[4] Shaowei Cai, Kaile Su, Chuan Luo, and Abdul Sattar. 2013. NuMVC: An efficient local search algorithm for minimum vertex cover. *Journal of Artificial Intelligence Research* 46 (2013), 687–716.

[5] François Delbot and Christian Laforest. 2010. Analytical and Experimental Comparison of Six Algorithms for the Vertex Cover Problem. *ACM J. Exp. Algorithmics* 15, Article 1.4 (nov 2010), 27 pages. https://doi.org/10.1145/1671970.1865971

[6] Magnús Halldórsson and Jaikumar Radhakrishnan. 1997. Greed is Good: Approximating Independent Sets in Sparse and Bounded-Degree Graphs. *Algorithmica* 18 (05 1997), 145–163. https://doi.org/10.1007/BF02523693

[7] Eran Halperin. 2002. Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. *SIAM J. Comput.* 31, 5 (2002), 1608–1623.

[8] Dorit S Hochbaum. 1983. Efficient bounds for the stable set, vertex cover and set packing problems. *Discrete Applied Mathematics* 6, 3 (1983), 243–254.

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

[10] Richard M Karp. 1972. Reducibility among combinatorial problems. In *Complexity of computer computations*. Springer, 85–103.

[11] Burkhard Monien and Ewald Speckenmeyer. 1985. Ramsey numbers and an approximation algorithm for the vertex cover problem. *Acta Informatica* 22, 1 (1985), 115–123.

[12] Silvia Richter, Malte Helmert, and Charles Gretton. 2007. A stochastic local search approach to vertex cover. In *annual conference on artificial intelligence*. Springer, 412–426.

Received 4 December 2022

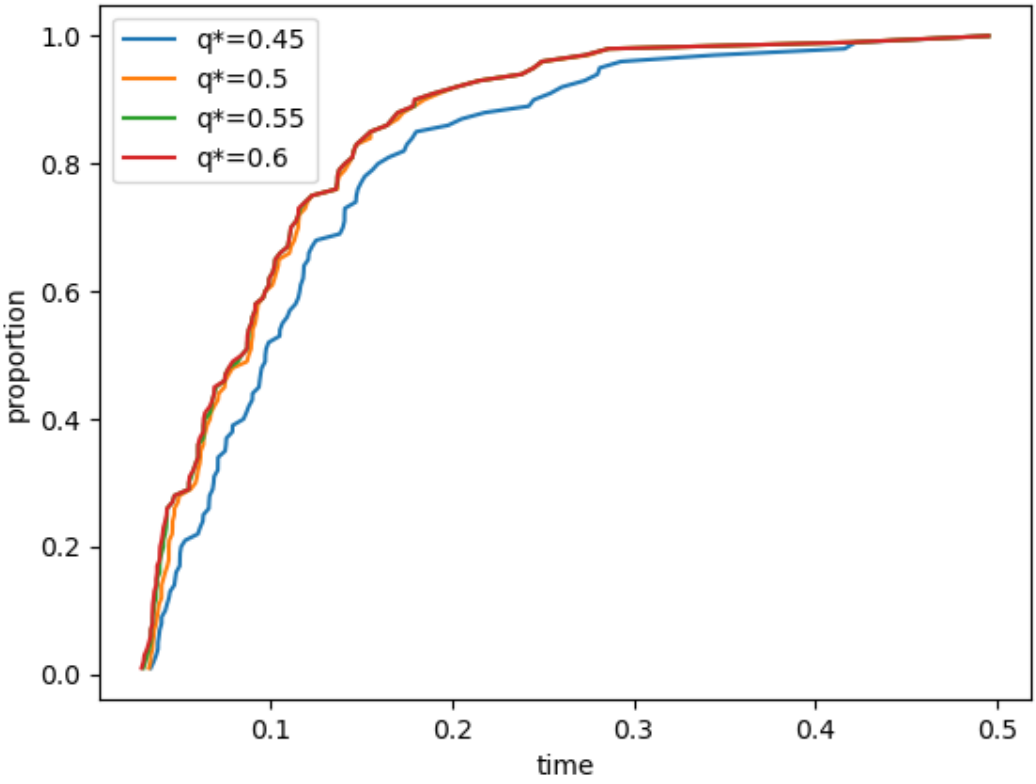

Fig. 2. QRTD of Power, Local Search 1

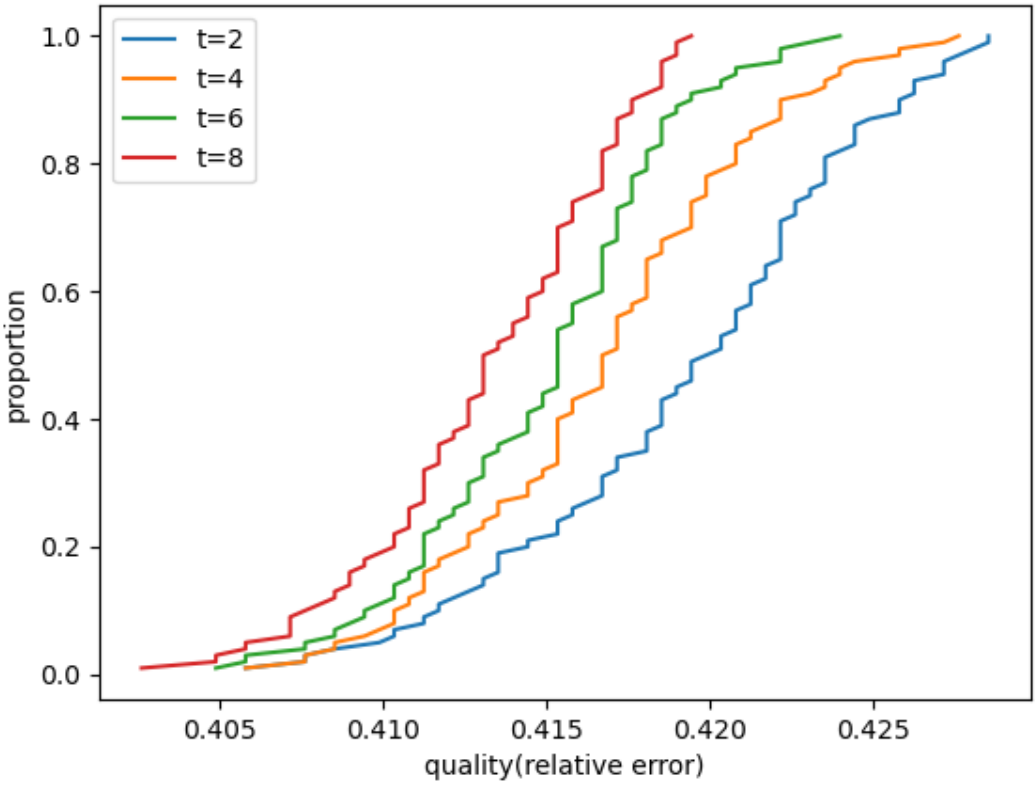

Fig. 3. SQD of Power, Local Search 1

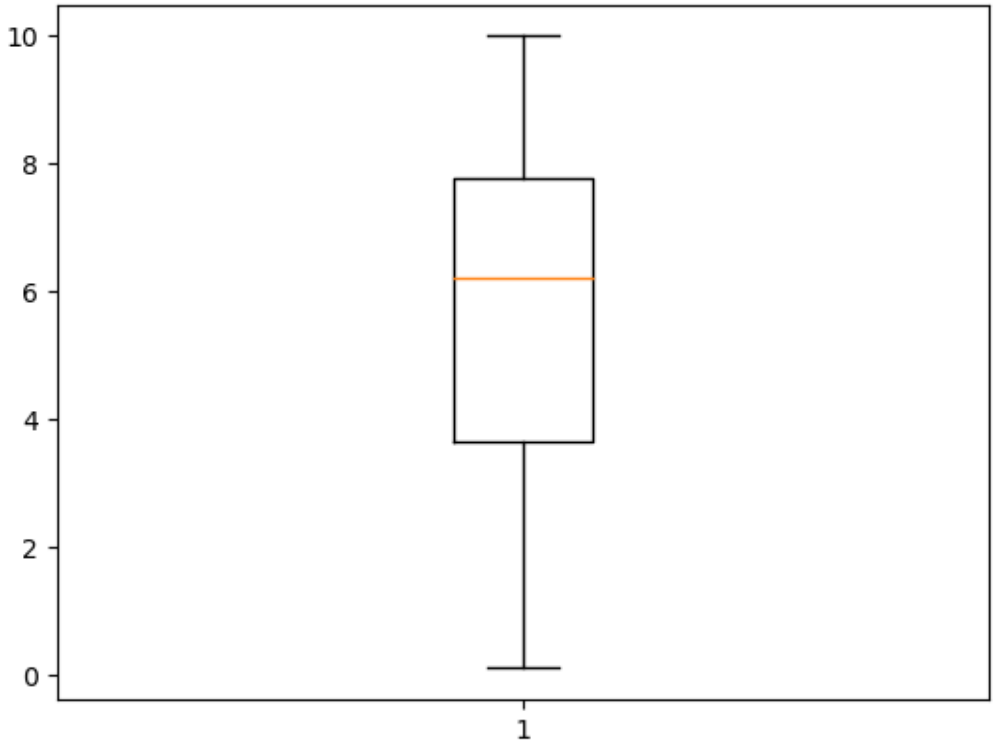

Fig. 4. boxplot of Power, Local Search 1

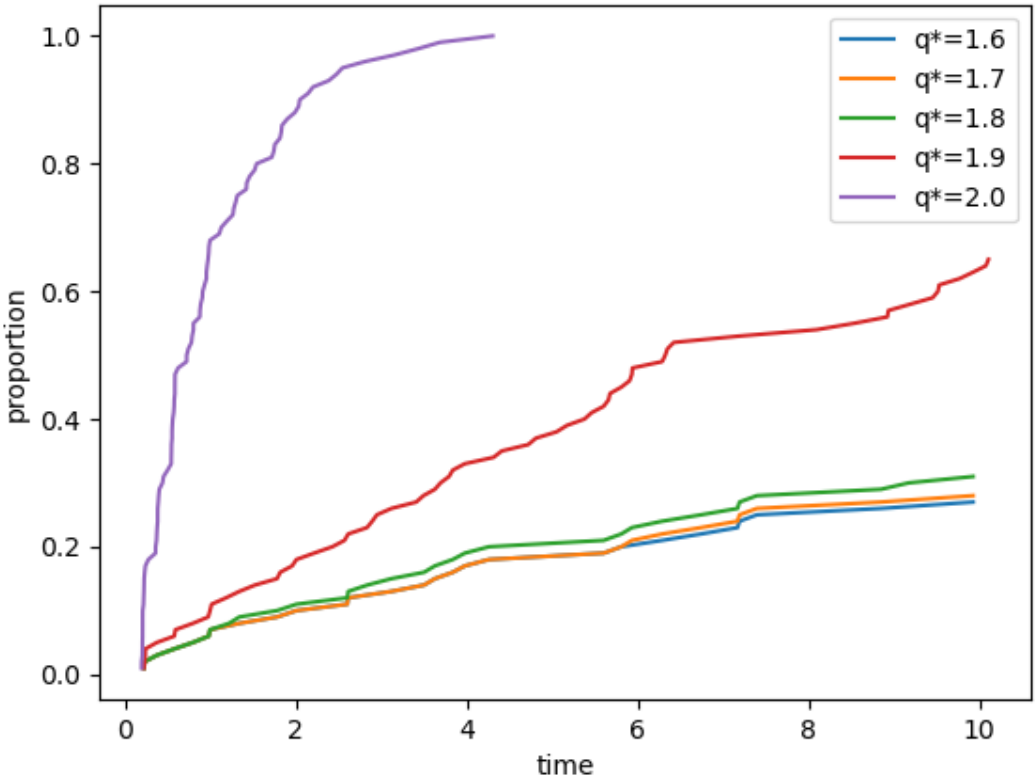

Fig. 5. QRTD of Star2, Local Search 1

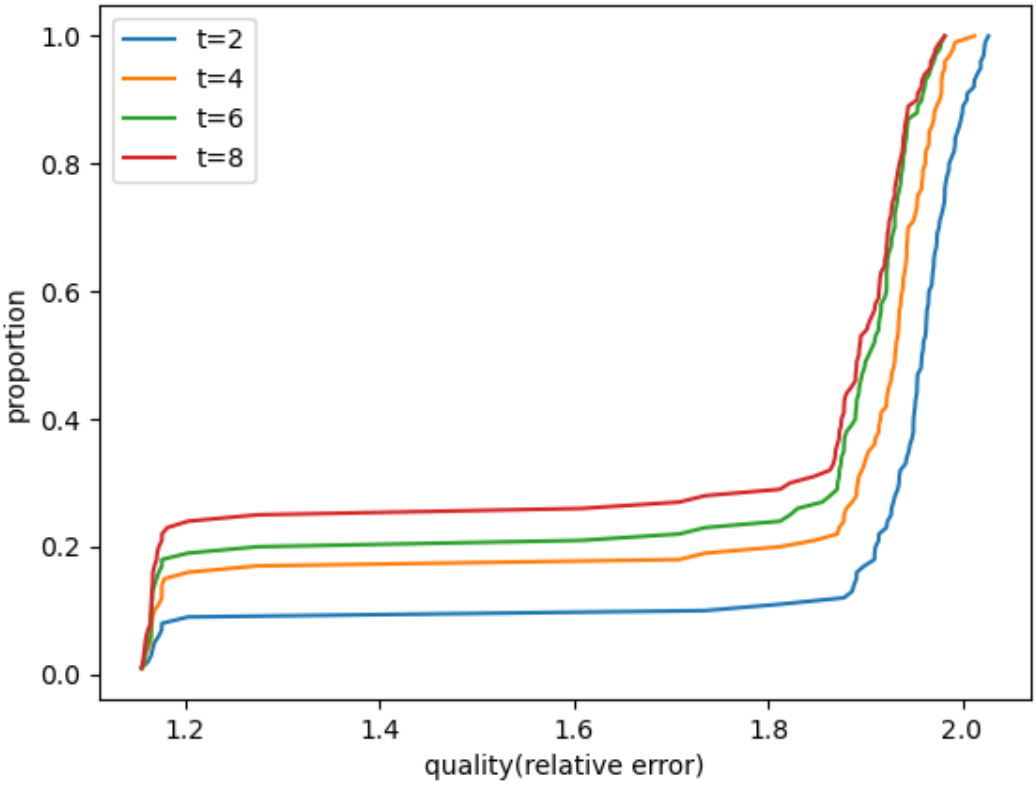

Fig. 6. SQD of Star2, Local Search 1

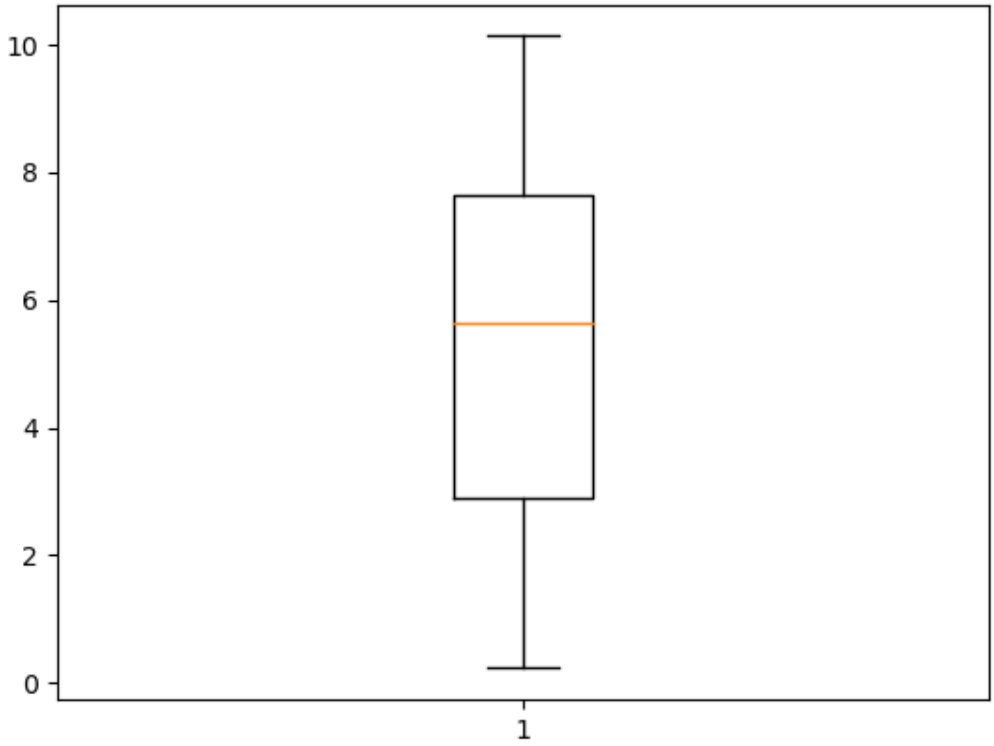

Fig. 7. boxplot of Star2, Local Search 1

Fig. 8. 1907 Franklin Model D roadster. Photograph by Harris & Ewing, Inc. [Public domain], via Wikimedia Commons. (https://goo.gl/VLCRBB).