CSE 6410 Project Progress Report

AMELIA GLAESE, YONGRUI LIN, and WANYANG GUO

1 INTRODUCTION

So far we have implemented 4 different algorithms to solve the minimum vertex cover problem: the exact Branch-and-Bound algorithm, an approximation algorithm using construction heuristics, and two different local search approximation algorithm, one using Iterated Local Search and the other using Simulated Annealing. In section 2 we describe the algorithms implemented and in section 3 we present preliminary performance results of the implemented algorithm. We intend to perform further analysis regarding runtime and may optimize our algorithms.

2 ALGORITHMS

2.1 Branch and Bound

The branch-and-bound algorithm begins with an empty set of vertices. The 2-approx is used to establish an initial crude upper bound. The algorithm selects the vertex with the highest degree of uncovered neighbors. The algorithm looks at this vertex and either adds the vertex to the partial solution, or leaves it out and adds its neighbors, depending on which is more promising. If a branch is determined not to be promising, it is not explored any further. Upon finding a vertex cover, the upper bound is updated to the size of the current best cover. The algorithm terminates when no promising branches remain.

More specifically, we have a graph $G = (V, E)$. The problem is to find a vertex cover with the smallest number of vertices in G.

SP: Given a partial solution C to the minimum vertex cover problem $G = (V, E)$, the resulting sub-problem is the minimum vertex cover problem for the subgraph $G' = (V', E')$, where $V' = V \setminus C$ and $E' = E \setminus \{\text{edges covered by } C\}$.

Choice: We can choose from the queue using the lower bound on the objective value for the subproblems.

Expand: We expand a subproblem by selecting the vertice $v_{maxDegree}$ with the highest degree and either adding it to the partial solution $C_{left} = C \cup v_{maxDegree}$ or adding its neighbors to the partial solution $C_{left} = C \cup \text{neighbors}(\text{v}_{\text{maxDegree}}, \text{E})$. Here adding all neighbors is the logical branch, because if $v_{maxDegree}$ is not added to our partial solution, all edges incident to $v_{maxDegree}$ can only be covered by its neighbors.

LB: We calculate the lower bound of a partial solution C by finding the maximal matching M of the subgraph i.e. $LB(C) = |C| + |M|$

Authors' address: Amelia Glaese; Yongrui Lin; Wanyang Guo.

ALGORITHM 1: Branch and Bound

Data: V, E
Result: $minCover$ = smallest vertex cover found
$C = V$
$LowerBound = 2Approx(V, E)$
$minCover$ = `MinVertexCoverBnB(`C, V, E`)`

def `MinVertexCoverBnB(`C, V, E`)`**:**
  if $|C| + getLowerBound(V, E) > LowerBound$**:**
    return $\emptyset$
  if C *is vertex cover*:
    $minC = C$
    $LowerBound = |minC|$
    return $minC$
  v = getMaxDegreeVertex(V, E)
  $neighborsV$ = neighbors(v, V, E)
  $C_{left} = C \cup v$
  $V_{left} = V \setminus v$
  $E_{left} = E \setminus$ edges(v, E)
  $minC_{left}$ = `MinVertexCoverBnB(`$C_{left}, V_{left}, E_{left}$`)`
  $C_{right} = C \cup \{neighborsV\}$
  $V_{right} = V \setminus \{v\}$
  $E_{right} = E \setminus \{neighborsV\}$
  $minC_{right}$ = `MinVertexCoverBnB(`$C_{right}, V_{right}, E_{right}$`)`
  if C_{left} *is empty or* $|C_{right}| \leq |C_{left}|$**:**
1   **return** C_{right}
  else:
2   **return** C_{left}

2.2 Construction Heuristics

For the construction heuristics, we used a Greedy Edge Deletion Algorithm. This algorithm returns the vertices of a maximal (but not necessarily maximum) matching. It's worse-case approximation ratio is 2 (proved in the class). More precisely, the tight worst-case approximation ratio is asymptotic to min $[2, \frac{1}{1-\sqrt{1-\epsilon}}]$ for graphs with an average degree of at least $\in n$ and asymptotic to min $[2, \frac{1}{\epsilon}]$ for graphs with a minimum degree of at least $\in n$.
First, the algorithm randomly chooses a edge $e = (u, v) \in E$ from the initial edge set E, and adds both end nodes of the chosen edge to the minimum vertex cover set V_{MVC}. In the next step, it removes every edge incident to either u or v from the edge set E. This procedure is repeated until there are no edges in left $E = \emptyset$. The resulting *minCover* is the approximated minimum set cover.

ALGORITHM 2: Construction Heuristics with Edge Deletion

Data: V, E
Result: $minCover$ = smallest vertex cover found
1 $minCover = \emptyset$
2 **while** $E \neq \emptyset$**:**
3 select a random edge $e = (u, v) \in E$
4 $minCover = minCover \cup \{u, v\}$
5 $E = E \setminus \{$edges incident to u and $v\}$
7 **return** $minCover$;

2.3 Local Search

2.3.1 Initial Vertex Cover. The function to construct the initial vertex cover used by both local search algorithms starts with an empty set and iterates over all $e = (v_1, v_2) \in E$. If an edge yet uncovered by the current set of nodes C is encountered, the node $v_i \in e$ with the higher degree is added to the set.

ALGORITHM 3: Construct Initial Vertex Cover

Data: V, E
Result: C = constructed vertex cover, I = vertices not included in C
C, I = InitialVertexCover(V, E)
def `InitialVertexCover(`C, I, V, E`)`**:**
    $C = \emptyset$ $I = V$ **for** $e = (v_1, v_2) \in E$**:**
        if *e not covered by C*:
            $v_i = \max_{v_i}(degree(v_1), degree(v_2))$
            $C = C \cup v_i$ $I = I \setminus \{v_i\}$
    return *minC*

2.3.2 Iterated Local Search Approach. The first Local Search algorithm utilizes an Iterated Local Search approach. First, an initial vertex cover is constructed using the greedy algorithm described in section 2.3.1. Then the algorithm randomly picks k vertices (here k$k = 10$) from the cover and computes the loss for each node. The loss is defined as the number of edges uncovered by the remaining set of nodes if the vertex was removed from the vertex cover:

$$\text{loss}(v_i) = C \setminus \{v_i\}, i \in k$$

In each iteration the algorithm removes the vertex with the least loss among the k nodes. This process is repeated until the resulting set of vertices is not a vertex cover. Then the algorithm starts exchanging vertices in the current set of vertices for vertices currently not included in the set. First the algorithm randomly picks k vertices $\in C$ and removes the node with the least loss. Then the algorithm randomly picks k vertices from the set I of vertices not included in C and computes the gain for each vertice. The gain is defined as the number of edges that are additionally covered by C, if the vertex was added:

$$\text{gain}(v_i) = C \cup \{v_i\}, i \in k$$

ALGORITHM 4: LS1 - Iterated Local Search

Data: V, E
Result: *minCover* = smallest vertex cover found
$C, I = InitialVertexCover(V, E)$
minCover = `MinVertexCoverLS1(`C, I, V, E`)`

def `MinVertexCoverLS1(`C, I, V, E`)`**:**
    while *(elapsed time ≤ cut off time)*:
        while *(C is vertex cover)*:
            minC = C
            minV = minLoss(C, V, E, k)
            $C = C \setminus \{minV\}$
            $I = I \cup \{minV\}$
        minV = minLoss(C, V, E, k)
        $C = C \setminus \{minV\}$
        $I = I \cup \{minV\}$ *maxV* = maxGain(I, V, E, k)
        $C = C \cup \{maxV\}$
        $I = I \setminus \{maxV\}$
    return *minC*

2.3.3 Simulated Annealing. The second local search algorithm is implemented using a simulated annealing approach, minimizing an objective function similar to the one described in XXX. The objective value of a set of vertices $S \subseteq V, G = (V, E)$, is defined as:

$$O(S) = \text{number of edges} \in E \text{ uncovered} + |S|$$

Like in the first algorithm, an initial vertex cover is constructed using the algorithm described in section 2.3.1. Subsequently, a random node $v \in C$ random node is selected. If $C \setminus \{v\}$ is a vertex cover and its objective value is smaller than the objective value of the current best solution *minC*, v is removed from C and *minC* is updated. If the objective value of the solution after removing f v, $C \setminus \{v\}$, is greater than the objective value of the current solution C, the probability $P(v)$ that v is added to the current solution is computed as follows:

$$P_{remove}(v) = e^{-\frac{\Delta O * (1 - \text{imp}(v))}{Temp}}$$

where $Temp$ is the cooling coefficient depending on the remaining running time of the algorithm

$$Temp = \text{rem. time} * 100$$

and imp(v) is the importance of the vertice v in Graph $G = (V, E)$, defined as the degree of v normalized by the number of edges in the graph

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

Based on the computed probability, v is either removed or not removed from C. The same process is implemented for adding a node to the current solution. A random node $u \in I$ is selected. If If $C \cup \{u\}$ is a vertex cover and its objective value is smaller than the objective value of the current best solution *minC*, u is added to C and *minC* is updated. If the objective value of the solution after removing f v, $C \setminus \{v\}$, is greater than the objective value of the current solution C, the probability $P(v)$ that v is added to the current solution is computed as follows, where $Temp$ and imp(u) are computed as above.:

$$P_{add}(v) = e^{-\frac{\Delta O * (1 - \text{imp}(u))}{Temp}}$$

ALGORITHM 5: LS2 - Simulated Annealing

Data: V, E
Result: *minCover* = smallest vertex cover found
C, I = InitialVertexCover(V, E)
minC = `MinVertexCoverLS2(`C, I, V, E`)`

def `MinVertexCoverLS2(`C, I, V, E`)`**:**
  while *elapsed time* $\leq$ *cut off time*:
    if C *is vertex cover and better solution than minC*:
      $minC = C$
    $randV$ = random vertex $v \in I$
    if qual($C \cup \{randV\}$) > qual(C)**:**
      $C = C \cup \{randV\}$
      $I = I \setminus \{randV\}$
    else:
      $Temp$ = remaining time $*100$
      ΔQ = qual($C \cup \{randV\}$) − qual(C)
      $probV = e^{-\frac{\Delta Q * (1 + \text{imp}(randV))}{Temp}}$
      if $probV$ *is met*:
        `/* remove node */`
        $C = C \cup \{randV\}$
        $I = I \setminus \{randV\}$
    $randV$ = random vertex $v \in C$
    if qual($C \setminus \{randV\}$) > qual(C)**:**
      $C = C \setminus \{randV\}$
      $I = I \cup \{randV\}$
    else:
      $Temp$ = remaining time $*100$
      ΔQ = qual($C \setminus \{randV\}$) − qual(C)
      $probV = e^{-\frac{\Delta Q * (1 - \text{imp}(randV))}{Temp}}$
      if $probV$ *is met*:
        $C = C \setminus \{randV\}$
        $I = I \cup \{randV\}$
    if *new* C *is vertex cover and better solution than minC*:
      $minC = C$
  return *minC*

3 PRELIMINARY RESULTS

The Branch and Bound algorithm time described in section 2.1 will find the optimal solution of to any minimum vertex cover problem if there is no cut-off. However, this can take a long time. For graphs with more than 1000 edges, the time to get to the any solution better than the initial lower bound already exceeds 10 minutes. We plan to run the algorithm longer for the final report to come to meaningful results. In table 1, the instances for which the Branch-and-Bound algorithm was not able to find any solution within 10 min of computation are marked "NA".

The edge deletion algorithm described in section 2.2 is faster than the Branch-and-Bound algorithm. However, if the number of edges is very large, iterating over all edges $e \in E$ and checking whether e should be removed may take a long time. In this case, the run time may be well over 10 minutes, as is the case for the *star* instance. As can be seen in the table above, the solutions provided by the edge-deletion algorithm can have a very large relative error, especially for large instances.

Compared to the edge-deletion algorithm, the local search algorithm provides solutions much closer to the optimal solution than the edge-deletion algorithm. After 10 min of computation, the relative errors of the provided solutions are consistently much lower. For small instances the algorithms often reach the optimal solution, for larger instances, the resulting relative errors are still comparatively low. Overall, the Simulated annealing algorithm seems to perform better because the resulting vertex covers after 10 minute of running time are slightly smaller than for the Iterated Local Search approach.

	Branch-and-Bound			Construction Heuristics			LS1: Iterated Local Search			LS2: Simulated Annealing		
Dataset	Time (s)	VC Value	RelErr	Time (s)	VC Value	RelErr	Time (s)	VC Value	RelErr	Time (s)	VC Value	RelErr
as-22july06	600	NA	NA	141.31	5998	0.81	603.11	3382.91	0.02	601.93	3381.45	0.02
delaunay_n10	600	753	0.07	582.34	898	0.28	600.16	768.00	0.09	600.08	764.89	0.09
email	600	633	0.07	488.2	792	0.33	600.20	651.64	0.10	600.07	637.36	0.07
football	2.07	94	0	463.28	100	0.06	600.00	98.00	0.04	600.00	98.00	0.04
hep-th	600	NA	NA	642.49	5736	0.46	602.09	3995.91	0.02	601.48	3994.45	0.02
jazz	4.11	159	0.01	548.27	170	0.07	600.03	162.91	0.03	600.02	159.25	0.01
karate	0.01	14	0	14.22	14	0	600.00	14.00	0.00	600.00	14.00	0.00
netscience	50.24	967	0.07	89.88	1182	0.31	600.19	899.00	0.00	600.06	899.00	0.00
power	600	NA	NA	291.29	3574	0.62	601.10	2345.00	0.06	600.67	2339.00	0.06
star	600	NA	NA	681.96	10192	0.48	618.92	7453.73	0.08	608.31	7452.27	0.08
star2	600	NA	NA	149.98	6806	0.49	637.62	5074.73	0.12	622.61	5074.64	0.12

Table 1. Preliminary results for cut off time $t = 600s$