Midterm

Navneel Singhal

2018CS10360

Contents

1 Problem 1

1.1 Statement

1.2 Solution

2 Problem 2

2.1 Statement

Let x^* be an optimum solution to the linear relaxation of the integer program for minimum vertex cover in $G = (V, E)$. Suppose $x_i^* \in \{0, 1/2, 1\}, 1 \leq i \leq n$ where x_i is the variable associated with $v_i \in V$. Further assume that the vertices of G are colored with 4 colors such that every pair of adjacent vertices have different colors. How will you round the solution x^* to obtain a 1.5-approximation for the minimum vertex cover. Give an algorithm and analyse its guarantee.

2.2 Solution

Since the statement mentions that we are already given such a colouring, we will assume that the input additionally contains the colouring information for all vertices in the graph.

Without loss of generality, let the colours be red, blue, green, yellow. Our algorithm will be as follows:

1. Find a colour c with the most number of vertices coloured with that colour which have $x_i^* = \frac{1}{2}$ (i.e., c is a majority colour among vertices v_i which have $x_i^* = \frac{1}{2}$).
2. Initialize set S with the empty set.
3. For each vertex v_i in V, add v_i to S if and only either
   - $x_i^* = 1$, or
   - $x_i^* = \frac{1}{2}$ and the colour of v_i is not c

   (i.e., round an x_i^* with value $\frac{1}{2}$ to 1 if and only if the colour of v_i is not c, else round it down to 0).
4. Return S.

For the sake of convenience, we suppose that the colour c that was picked is blue.

Claim 2.1

S is a vertex-cover.

Proof. Suppose there is an edge which has not been covered. Then there exists an edge (v_i, v_j) such that after rounding the solution from x^* to $\overline{x}$, we have $\overline{x}_i + \overline{x}_j < 1$. Since both of these are integers, this can happen iff both $\overline{x}_i, \overline{x}_j$ are 0. In the case that either of x_i^* and x_j^* is 0, the other variable must have been 1, which couldn't have been rounded down. And if either of them is 1, it isn't rounded down anyway, so the inequality is still satisfied.

Hence, the only possibility is that both of x_i^*, x_j^* are $\frac{1}{2}$, and both of $\overline{x}_i, \overline{x}_j$ are both rounded down to 0. This is only possible if both v_i, v_j were blue, which is impossible since the condition on the colouring guarantees that v_i, v_j must have had opposite colours.

This is a contradiction, and hence, S is a valid vertex cover. □

Claim 2.2

$|S| \leq \frac{3}{2} \cdot \mathbf{OPT}$

Proof. Let the set of vertices with $x_i^* = 1$ be A, the set with $x_i^* = 0$ be B, and the set with $x_i^* = \frac{1}{2}$ be C. Let D be the subset of C which consists of vertices which have colour blue.

Note that $\mathbf{OPT} \geq \mathbf{OPT}_{LP} = |A| + \frac{1}{2}|C|$, since the LP is a relaxed version of the original problem.

Now note that since blue is a majority colour among vertices in C, it must have at least $\frac{1}{4}|C|$ vertices.

Now consider the following inequalities:

$$
\begin{aligned}
\mathbf{ALG} &= |A| + |C| - |D| \\
&\leq |A| + \frac{3}{4} \cdot |C| \\
&\leq \frac{3}{2} \cdot |A| + \frac{3}{4} \cdot |C| \\
&= \frac{3}{2} \cdot \left(|A| + \frac{1}{2} \cdot |C| \right) \\
&= \frac{3}{2} \cdot \mathbf{OPT}_{LP} \\
&\leq \frac{3}{2} \cdot \mathbf{OPT}
\end{aligned}
$$

This completes the proof. □

3 Problem 3

3.1 Statement

We are given a directed acyclic graph $G = (V, E)$, vertices $s, t \in V$, edge-costs $c : E \to \mathbb{R}^+$, edge-lengths $l : E \to \mathbb{R}^+$, and a length bound L. Give a full polynomial time approximation scheme (FPTAS) to find the minimum cost path from s to t of length at most L. Hint: First give a dynamic program to solve the problem assuming the edge-costs are integers.

3.2 Solution

Firstly, we give a dynamic program that works if all the edge weights c_i are integer weights.

function MinCostPath($G = (V, E)$, s, t, $n = |V|$, $m = |E|$, $c[1 \dots m]$, $l[1 \dots m]$, L)
  Let ord be a topologically sorted order of vertices in G
    ▷ i.e., $ord[i]$ denotes the i^{th} vertex in this topologically sorted order. We can maintain a permutation that gives us, given a vertex v, the index i such that $ord[i] = v$, and this can be constructed in linear time.
  Let $C \leftarrow \max_i c[i]$
  Let $minLen[1 \dots n][0 \dots Cn]$ be a matrix initialized to ∞
                                                ▷ For our purposes, initializing it to $L + 1$ works as well.
      ▷ $minLen[i][j]$ will represent the minimum length of a path from $ord[i]$ to k with cost at most j
  Let q be the index such that $ord[p] = t$.
  Let p be the index such that $ord[q] = s$.
  if $p > q$ **then**
    return failure
  end if
  for i from 0 to Cn **do**
    $minLen[q][i] \leftarrow 0$
  end for
  for i from $q - 1$ down to p **do**
    for j from 0 to Cn **do**
      if $j \neq 0$ **then**
        $minLen[i][j] \leftarrow minLen[i][j - 1]$
      end if
      for each edge $(ord[i], ord[k])$ (with id r) with cost $c[r] \leq j$ **do**
        $minLen[i][j] \leftarrow \min(minLen[i][j], minLen[k][j - c[r]] + l[r])$
      end for
    end for
  end for
  if $minLen[p][Cn] > L$ **then**
    return failure
  end if
  Let c' be the minimum index with $minLen[p][c'] \leq L$.
  Let l' be $minLen[p][c']$
  Let $path \leftarrow$ list consisting only of p
  while $p \neq q$ **do**
    Let (p, r) be an edge with cost f and length g, and $minLen[r][c' - f] = l' - g$.
    $c' \leftarrow c' - f$
    $l' \leftarrow l' - g$
    $p \leftarrow r$
    Append p to $path$
  end while
  return $path$
end function

Claim 3.1

This algorithm returns failure if there is no path with length $\leq L$, else it returns a minimum cost with length at most L.

Proof. Note that since ord is a topological ordering of the vertices, any edge joins a vertex v_i to v_j only if $i < j$.

Firstly, we claim that $minLen[i][j]$ does indeed satisfy the property in line 7, after the nested iteration for j in the corresponding iteration for i is completed.

For the case $i = q$, it is fairly straightforward, since the empty path has length L and all paths have non-negative lengths so we can't do better.

Now suppose this is true for all $i' > i$. We show that this is true for i as well. Now consider any path with cost x and length y, starting at $ord[i]$. Then there must be an edge from $ord[i]$ to some $ord[k]$ with index r and cost $c[r]$ and length $l[r]$ such that the cost of the path from $ord[k]$ to t is $x - c[r]$ and the length is $y - l[r]$.

Fix some j. Then this holds for each path with cost $\leq j$ as well. Moreover, we must have the cost of the remaining path be non-negative (since edge costs are non-negative integers). So, it is sufficient to fix an edge and consider the minimum length of a path from $ord[k]$ to t with cost at most $j - c[r]$ (since path lengths are additive), and then take the minimum over all edges coming out of $ord[i]$, which completes the proof of the claim.

Now suppose there is no path with length $\leq L$ between s and t. Then we have $minLen[p][c] > L$ for all $c \in [0 \dots Cn]$, so we return failure in this case.

Now suppose there is indeed such a path. Then the number of edges in the path are at most $n - 1$, so the path cost is bounded above by $C(n-1) < Cn$. Hence, by the claim, we have that the minimum c for which $minlen[p][c]$ is $\leq L$ is indeed the minimum cost which is required (and such a c exists since we have assumed that such a path exists, so $minLen[p][cost(path)] \leq L$ by the claim). The rest of the algorithm is just construction of the path from the $minLen$ values, which is a straightforward simulation of walking on the path. □

Note that this algorithm uses Cn as an upper bound on the cost of **OPT**, and there is implicitly a lower bound of 1 since weights belong to $\mathbb{R}^+$, and the only case where the answer is 0 is the case when $s = t$, which is trivial. We will try to iteratively find better and better lower bounds so that we can get a better algorithm.

Firstly note that we can change the order of computation of our dynamic program from row major to column major order, by noting that (i, j) is computed strictly after $(i - 1, *)$, and it depends only on smaller j. By allocating memory only for the columns we need (this can be done by swapping rows and columns and the coordinates in our algorithm), we will get another algorithm which would run in $O((n + m) \cdot \mathbf{OPT})$. Call this the **good** algorithm.

To reduce the upper bound, we will try to binary search on it (with the initial range being $[1, Cn]$, where C is the max cost edge length).

For the binary search, we will use the following approximate predicate for a given $0 < r < 1$, given an upper bound B to test:

1. Remove all edges with cost $> B$, and replace the cost of edge e with $\lfloor cost \cdot n/(Br) \rfloor$
2. For c in 0 to $\frac{n}{r}$:
   - Compute the $minLen$ values for cost $= c$ (this corresponds to a column) using the good algorithm.
   - If we have $minLen[p][c] \leq L$, return false.
3. Return true

Claim 3.2

If this predicate returns false, then $\mathbf{OPT} \leq (1 + r)B$, else $\mathbf{OPT} > B$.

Proof. Suppose we return false. Then we have a path in the rounded instance with cost c, and since the number of edges on the path can be at most n, and the maximum length we have shaved off an edge by rounding to be Br/n rounding to be , we have the path cost at most $B + n \cdot Br/n = B(1 + r)$.

If we return true, this means that even in the relaxed instance (after scaling and rounding), there is no solution with cost $\leq \frac{n}{r}$, i.e., in the relaxed instance where we scale back after rounding, there is no solution with cost $\leq \frac{n}{r} \cdot \frac{Br}{n} = B$, so even for the original problem, there is no solution with cost B, which completes the proof. □

Hence using this algorithm, we can choose $r = \frac{1}{\sqrt{2}}$, and run this algorithm while doing a binary search on $\log \mathbf{OPT}$, while the ratio between the lower bound and the upper bound is not 2. It will always lead to a reduction of the ratio between by at least a factor of $\sqrt{2}$ each time, i.e., a reduction of log of the upper bound by at least $\frac{1}{2}$ each time. The time taken by the predicate to run is $O(n)$, so we will end up with an $O(n \log(Cn))$ algorithm till we get an upper bound which is at most two times the lower bound.

Note that since $\log C$ is the size of C in the input, this is polynomial time in input.

Hence using this algorithm (which in turn depends on the good algorithm), we have finally found an upper bound which is at most twice **OPT** (since lower bound is $\leq$ **OPT**), and that too in polynomial time.
Now our FPTAS will be as follows:

1. Let C be an upper bound on the answer found from the algorithm (this is different from the definition of C used above).
2. Multiply all edge costs with $\frac{2n}{\varepsilon C}$, and round them up.
3. Run the algorithm on this modified instance (and in the algorithm, replace the Cn in the algorithm by $\frac{2n}{\varepsilon}$ that we have chosen, since this is now the upper bound on the optimal path cost), and return the output of this algorithm.

Claim 3.3

This is a $(1 + \varepsilon)$ approximation.

Proof. Consider the optimal solution (under the condition that a solution exists). Note that our algorithm also returns a solution, since we haven't changed L or the edge lengths, or the structure of the graph. Suppose the edge costs on the path were $x_1, \ldots, x_k$. Then in the modified instance, it corresponds to $\lceil \frac{2nx_1}{C\varepsilon} \rceil, \ldots \lceil \frac{2nx_k}{C\varepsilon} \rceil$. Suppose the edge costs on the path returned by our algorithm are $y_1, \ldots, y_K$. Then in the modified instance, the edge costs are $\lceil \frac{2ny_1}{C\varepsilon} \rceil, \ldots \lceil \frac{2ny_K}{C\varepsilon} \rceil$.

We then have the following:

$$\begin{aligned}
\sum_{i=1}^{K} y_i &\leq \frac{C\varepsilon}{2n} \cdot \sum_{i=1}^{K} \left\lceil \frac{2ny_i}{C\varepsilon} \right\rceil \\
&= \mathbf{OPT}_{modified} \\
&\leq \frac{C\varepsilon}{2n} \cdot \sum_{i=1}^{k} \left\lceil \frac{2nx_i}{C\varepsilon} \right\rceil \\
&\leq \frac{C\varepsilon}{2n} \cdot \sum_{i=1}^{k} \left(1 + \frac{2nx_i}{C\varepsilon} \right) \\
&= \mathbf{OPT} + \frac{kC\varepsilon}{2n} \\
&\leq \mathbf{OPT} + \frac{k \cdot \mathbf{OPT}}{n} \\
&\leq \mathbf{OPT} \cdot (1 + \varepsilon)
\end{aligned}$$

□

Claim 3.4

The running time of this algorithm is $O(n(n + m)/\varepsilon)$

Proof. Note that the time taken by the algorithm is $O((n + m) \cdot$ (upper bound on OPT for the instance passed to the algorithm)), which is $O((n + m) \cdot \frac{2n}{\varepsilon})$, and we are done. □

This shows that we indeed have a FPTAS for this final algorithm.

4 Problem 4

4.1 Statement

4.2 Solution

5 Problem 5

5.1 Statement

5.2 Solution