CSE 565: Algorithm Design and Analysis **Due 2022-11-18T23:59**

Problem Set 5 Solutions

Instructor: Dr. Antonio Blanca
TA: Jeremy Huang

Release Date: 2022-11-04

Notice: Type your answers using LaTeX and make sure to upload the answer file on Gradescope before the deadline. Recall that for any problem or part of a problem, you can use the "I'll take 20%" option. For more details and the instructions read the syllabus.

Problem 1. Max Cut

In the `MAXIMUM CUT` problem we are given an undirected graph $G = (V,E)$ with a weight $w(e)$ on each edge, and we wish to separate the vertices into two sets S and $V - S$ so that the total weight of the edges between the two sets is as large as possible. For each $S \subseteq V$ define $w(S)$ to be the sum of all $w(e)$ over all edges u, v such that $|S \cap \{u,v\}| = 1$. Obviously, `MAX CUT` is about maximizing $w(S)$ over all subsets of V . Consider the following local search algorithm for `MAX CUT`:

start with any $S \subseteq V$
while there is a subset $S' \subseteq V$ such that
$|(S' \cup S) - (S' \cap S)| = 1$ and $w(S') > w(S)$ do: set $S = S'$

(a) Show that this is an approximation algorithm for `MAX CUT` with ratio 2.
(b) What is the time complexity of this algorithm?

Solution

a) We first show that any solution S that the algorithm outputs must have the property that $w(S) \geq \frac{1}{2}\sum_{e \in E} w(e)$. Since the size of the maximum cut can at most be $\sum e \in E w(e)$, this guarantees an approximation ratio of 2.

For contradiction suppose that $w(S) < \frac{1}{2}\sum_{e \in E} w(e)$. For each vertex $v \in V$, define $c(v)$ as the total weight of edges incident on v that cross the cut and $w(v)$ as the sum of weights of all the edges incident to v. Then

$$\sum_{v \in V} c(v) = 2c(S) < \sum_{e \in E} w(e) = \frac{1}{2}\sum_{v \in V} w(v)$$

since we count each edge twice, once for each of its endpoints. Thus, there must be at least one vertex v such that $c(v) < \frac{1}{2}w(v)$. If $v \in S$, define S' as $S \backslash \{v\}$, else define S' as $S \cup \{v\}$. All the edges incident to v that were not crossing the cut, will now cross the cut (adding $w(v) - c(v)$), but the ones that were crossing earlier may not (removing $c(v)$),hence

$$w(S') \geq w(S) - c(v) + (w(v) - c(v)) = w(S) + w(v) - 2c(v) > w(S)$$

Hence, there exists an S', which differs from S only by one element and $w(S') > w(S)$. This means that the algorithm could not have stopped at S, which is a contradiction.

b) At each iteration, we take $O(n)$ time to find if an S' exists with the required properties (find a node to add or replace or remove). Also, at each step the value of the cut increases and hence the total number of iterations is at most $O(\sum_{e \in E} w(e))$. Thus the running time is $O(n\sum_{e \in E} w(e))$. This is polynomial in the special case when all the edge weights are 1, but not in general.

Problem 2. Tree Jumper

In the MINIMUM STEINER TREE problem, the input consists of: a complete graph $G = (V, E)$ with distances d_{uv} between all pairs of nodes; and a distinguished set of terminal nodes $V' \subseteq V$. The goal is to find a minimum-cost tree that includes the vertices V'. This tree may or may not include nodes in $V - V'$.

Suppose the distances in the input are a metric (recall the definition on textbook page 273). Show that an efficient ratio-2 approximation algorithm for MINIMUM STEINER TREE can be obtained by ignoring the nonterminal nodes and simply returning the minimum spanning tree on V'. (Hint: Recall our approximation algorithm for the TSP.)

Solution

Let T be the minimum Steiner tree having cost C. Then we can follow the shape of the Steiner tree to obtain a tour of cost $2C$ which passes through all the vertices in the Steiner tree. Let i, j, k be adjacent vertices in the path. Using the triangle inequality, we know that $d_{ik} \leq d_{ij} + d_{jk}$. Hence, we can "bypass" an intermediate vertex j and connect i and k directly if we want, without increasing the cost.

Using this trick, we can bypass all the vertices *not* in V' that are present in the path, and also the vertices of V' which are being visited twice. This gives a path of cost at most $2C$, which passes though all the vertices of V' exactly once. Hence, this path is a spanning tree for V' of cost $2C$. This implies $cost(MST) \leq 2C$, thus giving the desired approximation guarentee.

Problem 3. Partial Independence

Give a polynomial running time $\left(\frac{1}{d+1}\right)$-approximation algorithm for maximum independent problem when the input graph G in undirected and the degree of all vertices of G is at most $d \in \mathbb{N}$.

Solution

Initially, let G be the original graph and $I = \emptyset$. Repeat the process below until $G = \emptyset$.

```
Pick the node v with the smallest degree and let I = I ∪ {v}.
Delete v and all its neighbors from the graph.
Let G be the new graph.
```

Notice that I is an independent set by construction. At each step, I grows by one vertex and we delete at most $d+1$ vertices from the graph (since v has at most d neighbors). Hence there are at least $|V|/(d+1)$ iterations. Let K be the size of the maximum independent set. Then the previous argument implies that

$$|I| \geq \frac{|V|}{d+1} \geq \frac{K}{d+1}$$

Problem 4. Bad Packing

Consider the following bin packing problem: given n items of sizes $a_1, a_2, ..., a_n$, find a way to pack them into unit-sized bins so that the number of bins needed is minimized. Consider a greedy algorithm: process all items in an arbitrary order; for the current k-th item, if there exists a partially packed bin that can further fit it, then put it in that bin; otherwise, open a new bin and put the k-th item in it. Show that the approximation ratio of above greedy algorithm is at most 2. Design an instance such that above algorithm performs at least as bad as $5/3 \cdot OPT$.

Solution

After putting all of the n items into m bins, at least $m-1$ bins are more than half full. Because if there are two bins less than half full, the items in one bin must be put into another bin following the greedy algorithm. If m is odd, i.e., $(m-1)$ is even, then the optimal solution must use at least $(m-1)/2+1$ bins: $OPT \geq (m-1)/2+1$; if m is even, i.e., $(m-1)$ is odd, then the optimal solution must use at least $m/2$ bins: $OPT \geq m/2$. In any case, we can show that $m/OPT \geq 2$.

Counterexample: Suppose we have 6 items with size 0.15, then 6 items with size 0.34, and finally 6 items with size 0.51. Using the greedy algorithm, there will be 10 bins: 1 bin for the first 6 items, 3 bins for the following 6 items and 6 bins for the final 6 items. The optimal answer is 6 bins with each bin contain one 0.15 item, one 0.34 item and one 0.51 item. The ratio for this instance is 5/3.

Problem 5. Lightning Punch

Consider a minimal-weighted hitting set problem define as follows. We are given a set $U = \{u_1, u_2, ..., u_n\}$ and a collection $\{B_1, B_2, ..., B_m\}$ of subsets of U, i.e., B_j is a subset of U. Also, each element $u_i \in U$ has a weight $w_i \geq 0$. The problem is to find a hitting set $H \subset U$ such that the total weight of the elements in H is as small as possible, i.e., to minimize $\sum_{u_i \in H} w_i$. Note that we say H is a hitting set if $H \cap B_j \neq 0$ for any $1 \leq j \leq m$. Let $b = \max_{1 \leq i \leq m} |B_i|$ be the maximum size of any of the sets $\{B_1, B_2, ..., B_m\}$.

(a) Design a b-approximation algorithm for above problem using the LP + rounding schema.

(b) Design a b-approximation algorithm for above problem using the primal-dual schema.

Solution

(a)

Consider the ILP, using x_i to represent whether element a_i is in hitting set S or not:

$$\min \sum_{i=1}^{n} w_i x_i$$

s.t.

$$0 \leq x_i \leq 1, i = 1, 2, ..., n$$

$$\sum_{i: a_i \in B_j} x_i \geq 1, j = 1, 2, .., m$$

Let x be the solution of the LP-relaxation of this problem. Now define the set S to be all those elements where $x_i \geq 1/b$, i.e., $S = \{u_i | x_i \geq 1/b\}$. let x^* be the solution of this ILP problem and H be the min-weighted hitting set.
(1) S is a hitting set. The set B_j contains at most b elements, and the sum of all x_i where $a_i \in B_j$. Therefore some $x_i > 1/b$ for some $a_i \in B_j$. Hence, S intersects with B_j by at least a_i.
(2) The total weight of all elements in S is at most bw_{LP} : for each $a_i \in S$, we know that $x_i > 1/b$, i.e. $1 < bx_i$, therefore:

$$w(S) = \sum_{a_i \in S} w_i \leq \sum_{a_i \in S} w_i b x_i \leq b \sum_{i=1}^{n} w_i x_i \leq b \sum_{i=1}^{n} w_i x_i^* = b w(H)$$

Thus, we have a hitting set S that is a b-approximation of this problem.

(b)

Solution: LP:

$$\min \sum_{i=1}^{n} w_i x_i$$

$$\text{s.t.} \sum_{i:a_i \in B_j} x_i \geq 1, j = 1, \ldots, m; \quad x_i \geq 0, i = 1, \ldots, n$$

Dual LP:

$$\max \sum_{j=1}^{m} y_j$$

$$\text{s.t.} \sum_{j:a_i \in B_j} y_j \leq w_i, i = 1, \ldots, n; y_j \geq 0, j = 1, \ldots, m$$

Let S be the current hitting set, y be the dual and let B_j be a set that has not been hit. Increase y_j until

$$\sum_{j:a_i \in b_j} y_j = w_i$$

for some $a_i \in B_j$. Include a_i in S. Repeat until all sets are hit.

Problem 6. UU Max Cut

Given an undirected graph $G = (V, E)$, we seek a cut $(A, B = V \backslash A)$ such that the number of cut-edges, denoted as $|E(A,B)|$ where $E(A,B) := \{(u,v) \in E \mid u \in A, v \in B\}$, is maximized. Consider the following greedy algorithm for this problem: process all vertices in V following an arbitrary order $\{v_1, v_2, \cdots, v_n\}$, and for the current vertex v_k, put it in B if $|N_A(v_k)| \geq |N_B(v_k)|$ and in A otherwise, where $N_A(v_k)$ is defined as $\{v_i \mid i < k, v_i \in A, (v_i, v_k) \in E\}$, and $N_B(v_k) := \{v_i \mid i < k, v_i \in B, (v_i, v_k) \in E\}$. Intuitively, this algorithm assigns a vertex to the different side with the majority of its adjacent vertices being already assigned to. Prove that this algorithm is an approximation algorithm. You will need to guess the approximation ratio, prove it, and design a tight example.

Solution

The approximation is 2. Since this algorithm assigns a vertex to the different side with the majority of its adjacent vertices being already assigned to, whenever a new vertex v_k is processed, at least half of the new edges are cut-edges. In other words, when we process a new vertex v_k, we are seeing $(N_A(v_k) + N_B(v_k))$ new edges, i.e. the number of neighbors of v_k which are already seen. Besides, $\max(N_A(v_k), N_B(v_k))$ are cut edges, so $\frac{\max(N_A(v_k), N_B(v_k))}{N_A(v_k)+N_B(v_k)} \geq \frac{1}{2}$ of the new edges are cut edges. This variant is guaranteed throughout the algorithm. Therefore, at the end, at least half of the total edges are cut edges, i.e. Algo $(G(V,E)) \geq \frac{1}{2}|E|$, where we use Algo to denote this algorithm. Obviously, $OPT(G(V,E)) \leq |E|$, where OPT denotes the optimal algorithm, so $\frac{OPT(G(V,E))}{Algo(G(V,E))} \leq 2$.

Tight example: Given graph $G(V,E)$ where $V = \{v_1, v_2, v_3, \cdots, v_n\}$ and $E = \{(v_1, v_2)\} \cup \{(v_1, v_i), (v_2, v_i) \mid i = (3,4,5,\ldots,n)\}$. We pick the points in the order of $\{v_1, v_2, v_3, \cdots, v_n\}$. Then, we will have v_1 in B, v_2 in A, v_i in $B, i = (3,4,5,\cdots,n)$. Hence, the number of cut edges is $(n-2)+1 = n-1$. The max cut edge is $2 \cdot (n-2) = 2n-4$. The max is reached by putting v_1 and v_2 in A and all the other vertices in B. The limit of the ratio $\frac{OPT}{Algo}$ is $\lim_{n\to\infty} \frac{2n-4}{n-1} = 2$. This example shows above analyis is asymptotically tight.