ShanghaiTech University

CS240 Algorithm Design and Analysis

Fall 2021

Problem Set 5

Due: 23:59, Jan.7, 2022

1. Submit your solutions to Gradescope (www.gradescope.com).
2. In "Account Settings" of Gradescope, set your FULL NAME to your Chinese name and enter your STUDENT ID correctly.
3. If you want to submit a handwritten version, scan it clearly. Camscanner is recommended.
4. When submitting your homework, match each of your solution to the corresponding problem number.

Problem 1:

In the maximum coverage problem, we have m subsets of the set {1, 2, . . . n}, denoted $S_1, S_2, ...S_m$. We are given an integer k, and we want to choose k sets whose union is as large as possible. Give an efficient algorithm that finds k sets whose union has size at least (1 - 1/e) · OPT, where OPT is the maximum number of elements in the union of any k sets. In other words,

$$OPT = \max_{i_1, i_2, ...i_k} \left| \cup_{j=1}^{k} S_{i_j} \right|$$

.

Solution:

we choose sets greedily, each time adding the set that includes the most elements that are not covered by any sets included in our solution so far.

Let n_i be OPT - the number of elements covered by the first i sets we choose. Initially, we have $n_0 = OPT$ since we haven't choose set. The k sets forming OPT cover at least n_i elements that our solution does not cover, which means one of these sets covers at least ni/k elements our solution doesn't cover. The set we add in iteration i + 1 can only cover more new elements than this set, i.e. we add a set covering at least n_i/k elements. So we have $n_i+1 \leq n_i - n_i/k = n_i(1-1/k)$. In turn, we have $n_k \leq (1-1/k)^k OPT \leq \frac{1}{e} OPT$, or equivalently our solution covers at least (1-1/e)OPT elements.

Problem 2:

Given an undirected graph $G = (V, E)$, you need to remove a minimum number of edges so that there is no triangles can be found in the graph.

Give a 3 -approximation algorithm that runs in polynomial time and explain why it works. (That is, you should guarantee that the number of edges removed is less than three times the optimal with the goal achieved.)

Solution:

Executive summary: The basic idea is to repeatedly find triangles in the graph G, and remove all three edges from the triangle. Since any optimal algorithm has to remove at least one edge from the triangle, we conclude that it is a 3 -approximation algorithm.

Algorithm description: First, we need a sub-routine that, given a graph $G = (V, E)$, finds a triangle. For today's purposes, assume that the graph is

stored as an adjacency matrix, and consider the simple algorithm that examines every we $\begin{pmatrix} n \\ 3 \end{pmatrix}$ combinations of three nodes (u, v, w) and checks whether there are edges connecting them. In this way, it takes $O\left(n^3\right)$ time to find a triangle (or report that there are no triangles). Certainly further optimization is possible.

Second, the algorithm proceeds as follows: while there are any triangles left in the graph, execute the following steps: - Let (u, v, w) be a triangle in G. - Remove edges $(u, v), (v, w)$, and (u, w) from G.

Explain why it works: First, it is easy to see that when the algorithm completes, there are no triangles. It remains to show that the algorithm is a 3-approximation algorithm. Define a triangle-set of G to be a set of disjoint triangles in G, i.e., a set of triangles such that no two triangles share an edge. (Triangles may share a node, however.) Notice that the algorithm produces a triangle-set, since after each iteration of the loop, it removes the edges from the previous triangle from the graph. Moreover, if the algorithm produces triangle set S, then the number of edges removed by the algorithm is exactly $3|S|$.

Let OPT be the optimal set of edges to remove from the graph G to ensure that it is triangle-free. Notice that for any triangle-set S, we can conclude that $|OPT| \geq |S|$, since OPT has to remove at least one edge in each triangle in S. Putting this together with the previous claim, we conclude that the number of edges removed by the algorithm is $3|S| \leq 3|OPT|$, and hence the algorithm is a 3-approximation of optimal.

Problem 3:

Recall that a TSP tour is a cycle that visits all the vertices of a graph exactly once.

Analogously, a three-tour consists of three disjoint cycles C_1, C_2, C_3 such that each vertex appears in exactly one of the cycles exactly once. The cost of a 3-tour is the sum of the lengths of the edges on the three cycles. (Here a single vertex is considered a cycle of length 1, so one or more of the cycles C_1, C_2, C_3 can consist of a single vertex.)

Suppose you are given a complete graph on n vertices with distances $dist(\cdot, \cdot)$ between pairs of vertices. Assume that the distances satisfy the triangle inequality in that for each triple of vertices i, j, k,

$$dist(i, j) + dist(j, k) \geq disk(i, k)$$

Describe a 2-approximation algorithm for the minimum cost three-tour problem.

Solution:

Compute the Minimum Spanning Tree (MST) and delete the two heaviest edges to recover a forest with three trees $T1, T2, T3$. The rest of the algorithm is analogous to the TSP approximation algorithm, but on three trees $T1, T2, T3$. In words, perform a DFS on each of the three trees $T1, T2, T3$ to obtain a traversal $\Gamma_1, \Gamma_2, \Gamma_3$. Shortcut the traversals to not have any vertex appearing more than once. This yields a Three-Tour solution.

Problem 4:

There is a course in SIST allows students to choose a time for their final project presentation. Each student has one project and all students need to choose. There are 8 time periods every day and it totally has 10 days. How many students must be in this course to make the probability 50 percent that at-least two students choose the same time?(random choose)(tips: $e^x = 1 + x + \frac{x^2}{2!} + \ldots$)

Solution:

Let the probability that two students choose the same time in n have same time be P(same). P(Same) can be easily evaluated in terms of P(different) where P(different) is the probability that all of them have different time. P(same) = 1 – P(different) P(different) can be written as 1 x (1 - 1/80) x (1 - 2/80) x (1 - 3/80) x .... x (1 – (n-1)/80) The n'th person should have a time which is not same as any of the earlier considered (n-1) persons. The above expression can be approximated using Taylor's Series.

$$e^x = 1 + x + \frac{x^2}{2!} + \ldots$$

provides a first-order approximation for e^x for x << 1:

$$e^x \approx 1 + x$$

$$e^{\frac{-a}{80}} \approx 1 - \frac{a}{80}$$

$$1 \times (1 - 1/80) \times (1 - 2/80) \times (1 - 3/80) \times \ldots. \times (1 - (n-1)/80)$$

$$\approx 1 \times e^{\frac{-1}{80} \times e^{\frac{-2}{80}} \ldots \times e^{\frac{-(n-1)}{80}}}$$

$$\approx 1 \times e^{\frac{-(1+2+\ldots+(n-1))}{80}}$$

$$\approx 1 \times e^{\frac{-(n(n-1))/2}{80}}$$

$$p(same) = 1 - p(different) \approx 1 - e^{-n(n-1)/(2\times 80)}$$

$$n \approx \sqrt{2 \times 80 \ln\left(\frac{1}{1 - p(\text{ same })}\right)}$$

n = 11

Problem 5:

In a weighted undirected graph G, define the distance of two vertices a and b is the length of the shortest path between them. Define that D is the longest path in G, that is

$$D = \max_{a,b} \text{DIST}(a, b)$$

Given a weighted undirected graph G with $|e|$ edges and $|v|$ vertices, you need to give a 2-approximation algorithm to find D in $O(elog(v))$ time. (i.e. Find a pair of vertices a,b such that $\text{DIST}(a, b) \geq \frac{1}{2}D$). Suppose that the time complexity of Dijkstra's algorithm is $O(elog(v))$.

Hint: You can first prove that for any vertice a, $D \leq 2 \cdot \max_b \text{DIST}(a, b)$

Solution:

Let the vertex that maximizes the distance from s be t. Then we have for any u

$$\text{DIST}(s, u) \leq \text{DIST}(s, t).$$

Putting this into triangle inequality involving u, v, and s gives:

$$\text{DIST}(u, v) \leq \text{DIST}(s, u) + \text{DIST}(s, v) \leq 2\,\text{DIST}(s, t)$$

The algorithm is:

i. Pick an arbitrary vertex s.

ii. Compute distances from s to all vertices u by running Dijkstra's algorithm with s as the source vertex.

iii. Let t be the vertex that maximizes $\text{DIST}(s, u)$, return s and t.

The approximation factor of this algorithm follows from part b), while its running time follows from that of Dijkstra's algorithm ($O(m \log n)$), plus the cost of doing a linear scan on the distance values.

Problem 6:

Recall that 3-SAT asks whether a boolean formula in 3-conjunctive normal form is satisfiable by an assignment of truth values to the variables.
The Max-3-SAT variation is an optimization problem that seeks to maximize the number of conjunctive clauses evaluating to 1. We assume that no clause contains both a variable and its negation.
Please design an algorithm which is 8/7-approximation algorithm and prove why.

Solution:

Given a Max-3-SAT algorithm with n variables $x_1 ... x_n$ and m clauses, set each variable 0/1 randomly.
Define Y_i = Iclause i is satisfied
A clause is only unsatisfied if all three literals are 0, so Pr(clause i is not satisfied) $= (1/2)^3 = 1/8$. Thus, Pr(clause i is satisfied) = 7/8, which means $E[y_i] = 7/8$. $E[Y] = 7m/8$. So the approximation ratio is m/(7m/8) = 8/7.