CSC 373 - Algorithm Design, Analysis, and Complexity **Summer 2016**

Assignment 3: Due Tuesday August 2nd, 10PM

Please follow the instructions provided on the course website to submit your assignment. You may submit the assignments in pairs. Also, if you use **any** sources (textbooks, online notes, friends) please cite them for your own safety.
You can use those data-structures and algorithms discussed in CSC263 (e.g. merge-sort, heaps, etc.) and in the lectures by stating their name. You do not need to provide any explanation or pseudo-code for their implementation. You can also use their running time without proving them: for example, if you are using the merge-sort in your algorithm you can simply state that merge-sort's running time is $\mathcal{O}(n \log n)$.

Every time you are asked to design an efficient algorithm, you should provide both a short high level explanation of how your algorithm works in plain English, and the pseudo-code of your algorithm similar to what we've seen in class. State the running time of your algorithm with a brief argument supporting your claim. You must prove that your algorithm finds an optimal solution!

1 That CS degree will finally pay off!

The cops finally cracked down the mafia in your city at a gangsta' party! Everybody was there. The police has a file on each one of them, so they know who works with who, but it's a government department, and yeah... these files are not very organized.

In order to know if there is a big cell within the group operating together or not, you were hired to determine if there are k mafiosi who know each other.

Politely prove to your police department that this problem is NP-complete :(

[Sketch of] Solution:

This is just CLIQUE. Represent every mafioso with a vertex, let $\{v_1, \ldots, v_n\}$ be the set of all vertices, two vertices are adjacent if and only if their corresponding mafiosi know each other. Looking for your mafiosi is looking for a clique in the graph. CLRS has a proof to NP-c ompleteness of CLIQUE.

2 Holidays at last!

The semester is almost over, and you can't wait to plan your next big trip! Ideally, you want your itinerary to be: Start in Toronto, visit every city in your list once, come back to Toronto. Since you're a jet setter, you don't do connecting flights. Before booking any tickets, you decide to check whether you can actually make an itinerary that satisfies your constraints. Show that nope, no luck, your problem is NP-complete. Formally, given n cities (including Toronto), two cities are adjacent if and only if there is a direct flight between them. You want to see if there is a way to start a trip from Toronto, visit every city once, and come back to Toronto.

[Sketch of] Solution:

This is just Hamilton Cycle. Construct a graph as follows: For every city introduce a vertex v_i. Two vertices are adjacent if and only if there are direct flies between their corresponding cities. The problem now becomes:

Finding a Ham Cycle in this graph. CLRS has a proof of NP-completeness of HAM CYCLE.

3 It's the small changes...

You're going on a long camping trip. You have n items you can potentially take with you. Each item has a weight w_i, and is useful is some way. Suppose you're given a function that gives you the value of each item.

$$\forall i \in [n], w_i \in \mathbb{Z}^{\geq 0}, v_i \in \mathbb{Z}^{\geq 0}$$

Suppose your bag has capacity $c \in \mathbb{Z}^{\geq 0}$. Ideally you want to take all the items, but you can't since your bag is small. Your goal is to select a subset of the item whose total value is maximized, without exceeding the total weight constraint of your bag.

More formally, you want a subset S of $k \leq n$ items that maximizes $\sum_{i=1}^{k} v_i$, and has total weight $\sum_{i=1}^{k} w_i \leq c$.

Consider the following greedy algorithm:

- Sort the items by $\frac{v_i}{w_i}$ in non-increasing order.
- Find the smallest k such that $\sum_{i=1}^{k} w_i > c$.
- Construct a set $S = \{1, 2, \ldots, k-1\}$.
- if $v_k > \sum_{i=1}^{k-1} v_i$, return $\{k\}$, otherwise return S.

Show that the above algorithm gives a 2-approximation.

Solution:

The algorithm returns either the first $k-1$ items or the k^{th} item. Let x denote the first solution, and y the second (the k^{th} item) solution. Then the algorithm returns $\max\{val(x), val(y)\}$.

Let x^* be an optimal solution to this problem.

Suppose our solution is sub-optimal, then we must have some leftover space in our bag. Suppose we were allowed to take a fraction of an item to place in the bag (we're not, but let's assume we are). Let z^* be a corresponding optimal fractional solution. That is, suppose we were allowed to take $\alpha \cdot v_k$ where $\alpha \in [0, 1)$. By taking a maximum fraction to fill the bag, we would either match or exceed the optimal value. Then:

$$\begin{aligned} val(x^*) &\leq val(z^*) \\ &= val(x) + \alpha v_k \\ &\leq val(x) + val(y) \\ &\leq 2\max\{val(x), val(y)\} \end{aligned}$$

Therefore achieving a 2-approximation.

4 Ain't no question like a graph theory question!

Let $G(V, E, w)$ be an undirected edge weighted graph, where $w(e) \geq 0, \forall e \in E$. You are asked to remove a subset of edges of minimum weight such that the remaining graph has no triangles.

Give a 3-approximation algorithm to solve this problem. Explain your algorithm, prove that it is correct, and that it achieves a 3-approximation ratio.

Solution:

We solve this problem via an integer linear programming relaxation. First, we need an ILP that solves the problem. Then, we relax it to get an LP, and then we round the solution. First, the ILP is defined as follows, with m variables $\{x_e : e \in E\}$; one for each edge:

$$\begin{aligned} \min \sum_{e \in E} x_e \cdot w(e) & \\ x_{e_i} + x_{e_j} + x_{e_k} \geq 1 & \quad \forall (e_i, e_j, e_k) \text{ that form a triangle} \\ x_e \in \{0, 1\} & \quad \forall e \in E \end{aligned}$$

Notice that we can compute this ILP in polynomial time, since if there are m edges, we can enumerate all triple of edges in $\mathcal{O}(m^3)$ time (and check if they form a triangle). Thus this is an ILP with a polynomial number of variables and a polynomial number of constraints. It is easy to see that any solution to this IPL results in a triangle-free graph, and conversely that the optimal edge set D is a feasible solution to this ILP.

We relax the ILP to an LP by allowing x_e to be an arbitrary real number between 0 and 1. We then solve the LP using any polynomial-time LP solver. Finally, we round the solution as follows: For every $e \in E$, let y_e be the rounded variable, and if $x_e \geq 1/3$, we set $y_e = 1$ otherwise $y_e = 0$.

First we show that this algorithm is correct (i.e. this actually returns a triangle free graph). Notice that in any triangle (e_i, e_j, e_k), the three variable $x_{e_i} + x_{e_j} + x_{e_k}$ must sum to at least 1, according to the LP's constraint and hence at least one of the three variable must be at least $1/3$. Thus, no triangle remains after removing set D.

Next we show the algorithm does achieve a 3-approximation ratio. Let OPT be the cost of the optimal edge set to produce a triangle-free graph, i.e. IP_{OPT} is the cost of the solution found by the ILP. Let LP_{OPT} be the cost of the solution found by the LP. We know that $LP_{OPT} \leq IP_{OPT}$, since the LP is a relaxation of the ILP.

Since we used a threshold of $1/3$, in the "worst" case we have increased some variable x_e from values of at least $1/3$ to 1. That is, $y_e \leq 3x_e$. Summing over all variables, we have

$$\begin{aligned} \sum_{e \in E} y_e \cdot w(e) &\leq \sum_{e \in E} 3x_e \cdot w(e) \\ &\leq 3LP_{OPT} \\ &\leq 3IP_{OPT} \end{aligned}$$

Therefore achieving a 3-approximation ratio.