College of Computer & Information Science — Fall 2014
Northeastern University — Handout 26
CS7800: Advanced Algorithms — 6 December 2014

Sample Solution to Problem Set 6

1. ($4 \times 5 = 20$ points) NP-completeness

Problem 34-2 of text.

Answer:

(a) Polynomial time. If we have n_x coins worth x dollars and n_y coins worth y dollars, we can try all ways to break up the n_x coins into two parts and n_y coins into two parts, and check which of these yield an even break up. This takes time at most $O(n^2)$.

(b) Polynomial time. We present a greedy algorithm for the problem. We process coins in non-increasing order of their denominations; let these be labeled $c_n, c_{n-1}, \ldots, c_1$. We set $p = n$. We repeat the following step until p equals 0 or we return **No**.

- If there exists an i such that $\sum_{i \le j \le p-1} c_j = c_p$, then we give c_p to Bonnie, and c_i through c_{p-1} to Clyde. If no such i exists, then return **No**. Otherwise, we set p to $i - 1$.

If p reaches 0, then we have equally divided the coins among Bonnie and Clyde.

Clearly the algorithm is polynomial time. We argue that it is correct. If the algorithm never returns **No**, then the coins have been equally divided by design. So the only case to consider is where the algorithm returns **No**. In this case, in some iteration, we could not find an i such that $\sum_{i \le j \le p-1} c_j = c_p$. We claim that $\sum_{1 \le j \le p-1} c_j < c_p$. Otherwise, there exists k such that $\sum_{k \le j \le p-1} c_j > c_p$, while $\sum_{k+1 \le j \le p-1} c_j < c_p$. This implies that there exists an integer m such that $(m+1)c_k > c_p > mc_k$. This is a contradiction since c_p is a multiple of c_k.

Thus, we have established that if the algorithm returns **No**, $\sum_{1 \le j \le p-1} c_j < c_p$. We claim that in this case it is impossible to divide the coins equally among Bonnie and Clyde. For the sake of contradiction, suppose there was a way. Let s_B (resp., s_C) be the sum of the values of the coins of denomination at least c_p that are with Bonnie (resp., Clyde). By the design of our algorithm, $s_B + s_C$ is an odd multiple of c_p. So $|s_B - s_C|$ is at least c_p. But since $\sum_{1 \le j \le p-1} c_j < c_p$, the remaining coins cannot make up for this deficit even if they are entirely given to the person who has the lesser sum. This completes the proof of the claim.

(c) NP-complete. Each cheque amount corresponds to an arbitrary integer. One can reduce the PARTITION problem to this problem. For each element e of an instance of the PARTITION problem, we create a cheque in the amount of e dollars. There is a way to split the elements of the PARTITION instance evenly if and only if there is a way to split the cheques evenly.

(d) Again NP-complete in general. One can reduce the PARTITION problem to this problem. For each integer element e of an instance of the PARTITION problem, we create a cheque in the amount of $101 \cdot e$ dollars. Any splitting of the cheques so the difference in the amounts that Bonnie and Clyde have is at most 100 dollars is a perfectly even split. So there is a way to split the elements of the PARTITION instance evenly if and only if there is a way to split the cheques in such a manner that the difference in the amounts that Bonnie and Clyde have is at most 100 dollars.

2. (5 × 3 = 15 points) Maximum-weight spanning tree

Problem 35-6 of text.

(a) Let $G = (V, E)$ be given by $V = \{a, b, c, d\}$ and $E = \{(a, b), (b, c), (c, d)$ with the weights of (a, b), (b, c), and (c, d) to be 1, 2, and 3, respectively. In this case, S_G and T_G both equal E.

(b) Let $G = (V, E)$ be given by $V = \{a, b, c, d\}$ and $E = \{(a, b), (b, c), (c, d)$ with the weights of (a, b), (b, c), and (c, d) to be 2, 1, and 3, respectively. In this case, S_G equals $\{(a, b), (c, d)$, while T_G equals E.

(c) Let e be the max-weight edge adjacent to vertex u. We claim that e is in T_G. If not, then adding e to T_G creates a cycle C with e. Let e' be the other edge in C incident on u. We have $w(e) > w(e')$; so removing e' and adding e to T_G leads to a spanning tree of higher weight, a contradiction.

(d) Arbitrarily root T_G at a node r. For every vertex $v \neq r$, let e_v denote the edge from v to its parent in this rooted tree. For any vertex v, let m_v be the weight of the maximum weight edge incident to v. Then, we have

$$w(T_G) = \sum_{v \neq r} w(e_v) \leq \sum_{v} m_v \leq 2w(S_G).$$

(e) We compute S_G in polynomial time. Then, repeatedly consider other edges, adding them if they do not form any cycle, until we have a spanning tree. Clearly the total weight of this tree is at least the weight of S_G, which is at least half of the weight of the maximum spanning tree.

3. (5 × 3 = 15 points) NP-completeness and approximation algorithms

Let G be an directed graph with k start nodes s_1 through s_k and k end nodes t_1 through t_k.

(a) Give a reduction from 3-SAT to show that it is NP-hard to determine whether there exist k paths, the ith path from s_i to t_i, $1 \leq i \leq k$, such that no two paths share an edge.

Answer: We first show that it is NP-hard to determine whether there exist k paths, the ith path from s_i to t_i, $1 \leq i \leq k$, such that no two paths share a *vertex*; i.e., all these paths are vertex-disjoint. It is then easy to give another reduction to the *edge-disjoint* case.

Consider a 3SAT formula ϕ with n variables and m clauses. We set $k = n + m$ and construct the following graph G. For each variable v and each clause c in the formula, we have two vertices: a source s_v and destination t_v (respectively s_c and t_c). If v (resp., $\neg v$) is in the clause c, then we add a node x_{vc} (resp., y_{vc}). We now describe the edges. We add an edge from s_c (and from t_c) to its associated x and y vertices; that is, we have an edge from s_c (and from t_c) to every node of the form x_{vc} and to every node of the form y_{vc}. We number the clauses in arbitrary order. We have an edge from x_{vc} to $x_{vc'}$ if c' is the next clause after c in the order that contains v. Similarly, we have an edge from y_{vc} to $y_{vc'}$ if c' is the next clause after c in the order that contains $\neg v$. Finally, we add the following edges for each variable v. We have an edge from s_v to x_{vc} (resp., y_{vc}) if c is the first clause (in the order) that contains v (resp., $\neg v$); if there is no such clause c, then we connect sv directly to t_v. We also have an edge from x_{vc} (resp., y_{vc}) to t_v if c is the last clause (in the order) that contains v (resp., $\neg v$); if there is no such clause c, then we connect sv directly to t_v.

Clearly, the above reduction is poly-time. We now show that ϕ is satisfiable iff the constructed graph G has node-disjoint paths from s_v to t_v and s_c to t_c for all v and c. Suppose ϕ has a satisfying assignment. We present a collection of node-disjoint paths. If v is true, we set path P_v to be the path from s_v to t_v traversing all the vertices of the form y_{vc}; otherwise, we set path P_v to be the path from s_v to t_v traversing all the vertices of the form x_{vc}. For each clause c, there exists a variable v such that v (or $\neg v$) is in c and is set to be true (or false). In the former case, we set path P_c to be s_c to x_{vc} to t_c; in the latter case, we set path P_c to be s_c to y_{vc} to t_c. It is easy to verify that all the paths are vertex-disjoint.

Suppose we have a collection of vertex disjoint paths. Consider the path P_v from s_v to t_v for any variable v. The first edge has to be of the form s_v to x_{vc} or y_{vc} for some clause c. Suppose it is the former. The node x_{vc} has only three other adjacent edges: one to s_c, one to t_c, and another to a node of the form $x_{vc'}$ if c' is the next clause that contains v or to t_v if there is no such c'. Clearly, the path P_v cannot contain s_c to t_c, so it must proceed to $x_{vc'}$ or to t_v. Following this reasoning, we obtain that P_v starts from s_v and then proceeds through all nodes of the form x_{vc} (or of the form y_{vc}) and terminates in t_v. Similarly, a path from s_c to t_c can be shown to be a 2-hop path: s_c to a node of the form x_{vc} or y_{vc} to t_c. This collection of paths yields the following satisfying assignment: set v to true if P_v goes through the y-nodes and false, otherwise. The vertex-disjoint paths for the clauses justify that each clause is satisfied.

Finally, we reduce from the vertex-disjoint paths problem, over say graph G, to the edge-disjoint paths problem on graph G', as follows. We replace every node z in G by two nodes z_i and z_o in G'; if there is an edge (z, z') in G, then we have an edge (z_o, z'_i) in G'. Finally, we have an edge (z_i, z_o) for every z. It is easy to see that any set of vertex-disjoint paths in G correspond to edge-disjoint paths in G', and vice versa.

We next consider an optimization version of the above problem. Here, we allow paths to share edges, and define the *load* $\ell(e)$ on an edge e to be the number of s_i-t_i paths that use e. The optimization problem then is to determine a set of s_i-t_i paths that minimizes $\max_e \ell(e)$.

(b) Write an integer linear program for the above problem. (*Hint:* You can view each s_i-t_i path as a flow of unit 1 from s_i to t_i.)

Answer: Let $G = (V, E)$. We have a variable $f_i(u, v)$ for each i, $1 \le i \le k$, and $(u, v) \in E$. Let us consider the following linear program.

$$\begin{aligned} &\min L \\ \sum_i f_i(u,v) &\quad\le\quad L \text{ for each edge } (u,v) \\ \sum_{(u,v)\in E} f_i(u,v) &\quad=\quad \sum_{(v,u)\in E} f_i(v,u) \text{ for each } i, v \ne s_i, t_i \end{aligned}$$

$$\sum_{(s_i,u)\in E} f_i(s_i,u) - \sum_{(u,s_i)\in E} f_i(u,s_i) \quad = \quad 1 \text{ for all } i$$

$$f_i(u,v) \quad \geq \quad 0 \text{ for all } i \text{ and } (u,v) \in E.$$

If we add the integrality constraint $f_i(u,v) \in \{0,1\}$, then we obtain an integer linear program equivalent to the problem at hand.

(c) Develop a randomized rounding algorithm which proceeds as follows: Relax the integrality constraint, and solve the LP; Decompose each s_i-t_i flow into a set of paths (we have seen this in a POW in class); Use randomized rounding to select a path. Fill in the details.

Answer: Relax the integrality constraint, and solve the LP. Decompose each s_i-t_i flow into a set of paths (we have seen this in class). Thus, each s_i-t_i flow assigns a flow value in $[0,1]$ to at most $|E|$ paths from s_i to t_i; the sum of these flow values is exactly one. So we select a path from s_i to t_i according to the probability distribution given by the path flows.

The next step is to show that the above rounding algorithm will achieve a load within an $O(\log n)$-factor of the optimal with high probability, say at least $1 - 1/n$, where n is the number of nodes in the graph.

(d) Show that the expected load of an edge e is equal to the total flow on the edge e in the LP solution.

Answer: Consider edge e. For $1 \leq i \leq k$, suppose P_i denote the set of s_i-t_i paths in the flow decomposition that contain edge e, and let $F_i(p)$ denote the path flow value of any path p in P_i. Then, the probability that path p in P_i is selected as the s_i-t_i path is exactly $F_i(p)$. Therefore, the expected load of edge e is simply $\sum_i \sum_{p\in P_i} F_i(p)$. By flow decomposition and LP definition, this sum is exactly the total flow over the edge in the LP solution.

(e) Using Chernoff bounds (described below) show that with probability at least $1 - 1/n$, the load of the path collection is within an $O(\log n)$ factor of the optimal achievable load.

Chernoff bound: Let $X_1, X_2, \ldots, X_n$ be n independent random variables each taking a value of 0 or 1. Let X denote $\sum_i X_i$. Then, for any $\delta > 0$, we have the following.

$$\Pr\left[X > (1+\delta)E[X]\right] \leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{E[X]}.$$

(*Hint:* For each edge, use the Chernoff bound to place a very small upper bound on the probability that the load of the edge exceeds $O(\log n)$ times the expectation. Then use a union bound to argue that with high probability, the load on every edge is within an $O(\log n)$ factor of the optimal.)

Answer: Fix edge e. Let X_i denote the random variable that is 1 if the selected s_i-t_i path uses edge e, and 0 otherwise. Let $X = \sum_i X_i$. Then, X is the load on edge e. Let L denote the value of the LP solution. We set $\delta = (c\log n - 1)L/E[X]$ for a constant c that will be specified below. By Chernoff bound, we obtain

$$\Pr\left[X > (1+\delta)E[X]\right] \quad \leq \quad \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{E[X]}$$

$$
\begin{aligned}
&\leq \frac{e^{\delta E[X]}}{\delta^{\delta E[X]}} \\
&\leq \left(\frac{e}{(c\log n - 1)}\right)^{(c\log n - 1)L}.
\end{aligned}
$$

Since L is at least 1, we can set c sufficiently large so that the above is at most $1/n^3$. Since there are at most n^2 edges, it follows that the probability that any edge has load more than $Lc\log n$ is at most $n^2/n^3 = 1/n$.