Massachusetts Institute of Technology
6.854J/18.415J: Advanced Algorithms
David Karger

Handout 14
Wednesday, Obtober 19, 2005

Problem Set 6 Solutions

Problem 1.

(a) We will reduce the problem to a min-cost max flow in a negligibly larger network. In the beginning we determine the value of a maximum flow in an original network G. Denote this value by η.

Let s and t be, respectively, a source and a sink in G. We create a new node s', which is a new source, and add two directed arcs of cost 0 each, one from s' to s of capacity η, and the other from s to t of capacity $10\% \cdot \eta$. The first edge bounds the value of a flow in the modified network G', and the second allows up to 10% of a flow to be sent at no cost right to the sink.

We run any min-cost flow algorithm on G', and cutting a resulting flow function to the old network G, we achieve a flow in G of the required properties. Clearly, the value of the flow is greater than or equal to $90\% \cdot \eta$, and each value of flow from $90\% \cdot \eta$ to $100\% \cdot \eta$ is possible.

The running time of a min-cost flow algorithm dominates the running time of the whole algorithm, so the asymptotic complexity of the whole algorithm equals the asymptotic complexity of the min-cost flow algorithm on G'.

(b) A slight modification of the solution to part (a) works. This time, the additional arc from s to t has capacity ∞ and cost K. The flow that is not sent through G must be sent along the additional arc in G', and therefore, costs K times its value.

Problem 2.

(a) All reduced costs of arcs in a residual network are nonnegative, and if we modify the cost of an arc (v, w) by one, at most one of arcs, (v, w) or (w, v), will have a negative cost of -1 in the residual network. Assume w.l.o.g. that an arc (v, w) of a capacity u in the residual network has cost -1. Since all other arcs have nonnegative costs, if there is a cycle of negative cost, it goes through this arc and all its other arcs have cost 0. Therefore, in the residual network, we try to push as much flow as we can, but no more than u units, from v to w over arcs of cost 0, and we send the same amount of flow from v to w. Assuming we get a flow of value f, then the cost of the total flow decreases by $-uf$.

If arc (v, w) becomes saturated, there is no arc of negative cost in the modified residual network, and we are done. Otherwise, we increase by 1 the potential of each vertex x from which there is a path along arcs of cost 0 to v in the new residual network. The potential of w will not increase, because this would mean that the flow from w to v over arcs of cost 0

has not been maximal, and this will make the reduced cost of the arc from v to w increase to 0. Only the reduced cost of an arc incoming to a vertex whose potential have increased may decrease, and this will happen only if the arc comes from the vertex whose potential have not increased. In such case the reduced cost of the arc was positive before the modification, and if decreases by 1 it stays nonnegative.

We can modify the potential function, to get a feasible potential, so the cost of the flow after the modification is for sure optimal.

(b) We start with some maximum flow, zero costs, and a potential function equal to 0 for each vertex. There are $O(\log C)$ turns, in each we shift in consecutive bits of costs. At the beginning of a turn we double costs of each arc, and potential of each vertex. It does not affect the optimality of a current flow, and both the potential function and reduced costs modified in this way keep their properties. Next, for each arc (p, q) we add the next bit of cost c_{pq} of (p, q), if it is set. This increases or decreases the cost of arc (p, q) by 1, depending on the sign of c_{pq}. If there appears an arc of reduced cost -1 in the residual network, we call the algorithm in part (a) to repair the current flow.

The total number of calls to the algorithm in part (a) is upper-bounded by $O(m \log C)$.

Problem 3.

Focus on a single inequality

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n \geq b,$$

describing a half-space S. We can assume that at least one a_i is non-zero. A ball of radius r fits inside the half-space if and only if its center belongs to the half-space, and is at distance at least r from the boundary hyperplane P described by the equation

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n = b.$$

The vector $\vec{v} = [a_1, a_2, \ldots, a_n]$ is perpendicular to hyperplane P, and oriented into the inside of S. Centers of balls of radius r that fit inside S are points of the halfspace S' which is the translation of P by r into the direction pointed by $\vec{v}$. Let $\vec{w} = [b_1, b_2, \ldots, b_n]$ be the normalized vector of v. S' is the translation of S by $r\vec{w}$, and

$$a_1(x_1 - rb_1) + a_2(x_2 - rb_2) + \cdots + a_n(x_n - rb_n) \geq b$$

is an equation that describes S'. The equation is a linear equation in x_1, x_2, ..., x_n, and r.

Come back to the problem that we need to solve, introduce an additional variable r, and modify all equations in the way that we have just presented. We achieve a linear program of $n + 1$ variables, and maximizing value of r, we find the center $(x_1, x_2, \ldots, x_n)$ of a ball of the greatest radius.

Problem 4.

(a) We create the following flow network. The set of vertices consists of a source s, a sink t, a vertex s_g for each graduate student g, and a vertex o_i for each office i. We add a directed arc of unit capacity and cost 0 from s to each s_g. For each office i we add a directed arc from o_i to t of cost 0 and capacity n_i. These arcs designate that a student wants to be assigned to one desk, and bound the number of students in an office. Now, for each student g and each office i we create an arc from s_g to o_i of capacity 1 and cost $-p_{ig}$—the arc describes that g wants to pay p_{ig} for assignment to i.

For each assignment of students to desks there is a corresponding flow. A solution to the problem does not exist if the value of a maximum flow in the network is less than the number of students. Conversely, if the value of a maximum flow equals the number of students, there is an integral flow (see part (b)), and a flow of unit value along some arc from s_g to o_i says that student g is assigned to office i. Minimizing the value of the maximum flow, we maximize the amount of money that we get, and see part (b) that the minimum is achievable by some integral flow.

(b) Since the capacities of all the arcs are integral, using arbitrary algorithm presented in class we get an integral flow, in which students are assigned a single desk, and they do not have to share it.

(c) We create a **nonnegative** variable x_{ig} for each office i and each student g. Next, for each office i we add a constraint

$$\sum_g x_{ig} \leq n_i,$$

and for each student g

$$\sum_i x_{ig} = 1.$$

Maximizing the value of

$$\sum_g \sum_i p_{ig} x_{ig},$$

we achieve a linear program that expresses the problem of finding a flow of the minimum cost and of the value equal to the number of students in part (a), where x_{ig} represents the value of flow from s_g to o_i.

Problem 5.

(a) We can think of our situation almost as of a flow, where nodes are currencies, and arcs are pending orders going from a currency a client wants to take to a currency they want to give us instead. The only difference from the standard flow problem is that the value of a flow entering an arc not necessarily equals the value of the same flow leaving the arc—it is multiplied by r_i for an order of a client i.

Let us express this situation as a linear program. We create a variable x_i for each order i. The variable describes the amount that is being converted, or the value of flow entering an arc corresponding to this order. In the beginning we bound a flow along this arc:

$$0 \leq x_i \leq u_i.$$

Next, we bound amounts that we can spend—first the amount of dollars:

$$\sum_{a_i=\$} x_i \leq \sum_{b_i=\$} r_i x_i + D,$$

and then the amount of every other currency c:

$$\sum_{a_i=c} x_i \leq \sum_{b_i=c} r_i x_i.$$

We maximize the amount of yens that we get, that is the value of sum

$$\sum_{b_i=¥} r_i x_i - \sum_{a_i=¥} x_i.$$

Now we will discuss features of solutions to this linear program. First of all, it solves a problem under the assumption that we can exchange money with a client arbitrary number of times (even $\aleph_0$ times). We can decompose a flow it describes into a set of (at most m where m is the number of orders) simple paths and paths ending with an infinite cycle (those cycles resemble black holes, since money get in, circulate, and never get out). Furthermore, all the paths start at dollars. Why this is true? This can be proven by induction. Assume that we want to decompose a flow containing non-zero flow along m arcs. We start at dollars, and randomly traverse a network along arcs over which flow is non-zero. Eventually, we get either to a node from which there is no way out, or to a node which is on some cycle. In the latter case we loop on this cycle, and the value of flow on each arc of the cycle decreases by some fixed factor less then 1 in each visit, and the total value of flow over each arc on the cycle is a limit of some geometric sequence. In both cases, when we finish in a cycle or in some vertex, we can set the initial amount of dollars so the flow over our path equals the flow over some arc e, and if we subtract flow over our path from the flow in the network, e will "disappear", and there will be at most $m-1$ arcs of non-zero flow. Moreover, there are is no flow that is separate from the source, consisting simply of cycles, this would violet either the assumption that $\prod r_i < 1$ for cycles, or the constraint that we cannot spend more than we get in some currency.

(b) Note that it does not pay to have cycles in a flow. They constitute just mentioned black holes, that simply consume our money. Actually, we can get rid of cycles in the flow. It will be shown in part (c) how to do it. Suppose then that there are no cycles in the flow. If there are no cycles, then there is an ordering of edges, and we can respond to each pending

order at most once, never having to borrow money. This can be done by consideration of currencies in some topological order determined by the flow.

(c) We will get rid of infinite paths and finite paths that do not end at yens (these money are "lost"), and possibly find a flow that consumes even less dollars.

First, we need to determine the maximum amount C of yens that we can get. Then we add additional constraint that an amount we get is optimal:

$$\sum_{b_i=¥} r_i x_i - \sum_{a_i=¥} x_i = C,$$

and minimize the amount of dollars that is spent:

$$\sum_{a_i=\$} x_i - \sum_{b_i=\$} r_i x_i.$$

In a decomposition of a resulting flow there will be no finite paths ending in a currency different than yens, and there will be no paths ending with cycles, since getting rid of them, we would get a better solution to our linear optimization program, spending less dollars. This implies as well that the modified linear program produces a solution in which we end day only with dollars and yens, and the amount of yens is optimal.