CMSC 27200 Assignment 6

Rachel Hwang

February 19, 2014

- **Exercise 29.1-4 (Cormen)**
  To convert this into standard form, we take the following steps

  1. Since the given is a minimization problem, negate all coefficients in the objective form $2x_1+7x_2+x_3 \rightarrow -2x_1 - 7x_2 - x_3$.
  2. One of our zero constraints is less than or equal to zero, which we can remove by replacing x_3 with two variables that satisfy a ≥ 0 constraint as follows: $x_3 = x_3' - x_3'' \leq 0$
  3. Next, we have a variable not subject to a zero constraint. We amend this by replacing each occurance of x_1 with $a_{i1}x_1 = a_{i1}x_1' - a_{i1}x_1''$.
  4. Remove equality constraint by replacing it with two inequality constraints $x_1 - x_3 = 7 \rightarrow (x_1 - x_3 \geq 7), (x_1 - x_3 \leq 7)$
  5. Finally we remove the greater-than-or-equal-to inequalities by multiplying by -1.

  This gives us:
  Maximize $-2x_1' + 2x_1'' - 7x_2 - x_3' + x_3''$
  Subject to:

$$\begin{aligned} x_1' - x_1'' - x_3' + x_3'' &\leq 7 \\ -x_1' + x_1'' + x_3' - x_3'' &\leq -7 \\ -3x_1' + 3x_1'' - x_2 &\leq -24 \\ x_3' - x_3'' &\leq 0 \\ x_1', x_1'', x_2, x_3', x_3'' &\geq 0 \end{aligned}$$

- **Exercise 29.1-7 (Cormen)**
  We can negate the first two constraints as follows:

$$\begin{aligned} x_1 - x_2 &\geq 1 \\ x_1 + 2x_2 &\geq 2 \end{aligned}$$

  Given these two constraints, there is no limit on how large x_1 can be. For any values of x_1, x_2 you can choose which satisfies the constraints, you can always choose a larger value of x_1 which will better maximize the objective function. For example, we can define an infinite sequence of ordered pairs of x_1 and x_2 values, $f(i) = (2(i+1), 1)$. For any $i \geq 0$, the output values of this function satisfy all program constraints. If we define our objective function to be $g(x_1, x_2)$, then the limit of $g(f(i))$ is infinity, thus the linear program is unbounded.

- **Exercise 7.13**
  Let our node capacity graph be $G(E, V)$. We can convert this into an ordinary max-flow problem by artificially inserting each node's capacity cost into a new graph $G'(E', V')$. For our node-capacity graph G, we split each node v with capacity c into two nodes in V', v' and v'', joined by an edge $e'_{v',v''}$ bearing the capacity of v. That new edge will be part of a special set $E*$. v' will inherit all incoming edges to the original node (for any $e_{u,v}$ where $u \in V$, insert $e'_{u,v'}$ with infinite capacity) and v'' will inherit all outgoing edges (for any $e_{v,u}$ where $u \in V$, insert $e'_{v',u}$ with infinite capacity). We have now set up a one-to-one correspondence where each node's capacity can now run the ordinary max flow algorithm as usual. Building G' will run in $O(|V| + |E|)$ (constant work for each edge and constant work each node). Since Ford-fulkerson runs in polynomial time and the build runs in polynomial time, our overall algorithm runs in polynomial time.

We can prove that the max-flow found in G' is the same as the max flow of G by contradiction. Assume that $maxflow(G) > maxflow(G')$. Since we have a one-to-one method of mapping capacities from G to G', we must be able to find a better flow using the same capacities in G'. This contradicts the fact that we began with $maxflow(G')$. If we assume $maxflow(G) < maxflow(G')$, then the max flow of G' must use some set of capacities that are not available in G, which is impossible since G' is constructed from G by direct corespondence.

A cut in G can be defined as dividing the set of nodes into two sets A and B, one containing s and the other t. The cut cost is equal to $\sum_{v \in B} C_v$, where $\exists e_{u,v}$ for any $u \in A$. Using the same logic as in the above proof that $maxflow(G) = maxflow(G')$, we know that $mincut(G) = mincut(G')$.

- **Exercise 7.25**
  Consider each person as a node in a directed graph $G(V, E)$ where each person is $v_i \in V$ and each debt is some $e_{v,w} \in E$ where v owes w money equal to cost of the edge. If $|E| < n$, we know that we can settle all debts in $n - 1$ checks or less; each person simply writes a check for each debts they owe and recieves a check for each debt they are owed. This will obviously balance all debts.

  Now, if $|E| \geq n$, we know that there must be some set of nodes which are connected creating a cycle (ignoring directedness for a moment), since trees are maximally acyclic and contain $n - 1$ edges.

  For any cycle in G, we can remove an edge in that cycle while preserving imbalance at each node. Let the vertices in the cycle be $v_1, v_2, \ldots, v_k, v_1$. We choose the edge with the smallest cost e', WLOG let's say $e = e_{v_1,v_2}$ from the cycle and remove it. For each of the other edges in the cycle, if the edge is directed in the same direction as e' (in this case v number increasing), subtract $C_{e'}$ from the cost of that edge. If the edge is in the opposite direction, add the cost instead. We then remove any edge costs equal to 0. Note that since we are only ever subtracting the minimum capacity value, no edge cost will ever fall below zero.

  Now, we can show that this method will always preserve imbalances for each person. Consider some v_i and its two neighbors v_{i-1} and v_{i+1} in a cycle. One of the following cases must hold

  - v_i owes one neighbor money and is owed money by the other neighbor ((v_{i-1}, v_i) and (v_i, v_{i+1}) OR (v_i, v_{i-1}) and (v_{i+1}, v_i)). Both edges will decrease or increase by precisely the same amount, so balance is preserved.
  - v_i owes both neighbors money ($(v_i, v_{i+1}$ AND $v_i, v_{i-1})$. One edge will decrease and the other will increase by the same amount, so balance is preserved.
  - v_i is owed by both neighbors. Like above, one edge will decrease and the other will increase by the same amount, so balance is preserved.

  Because we have shown that imbalances are always preserved and debts never change direction (because as explained, debts always remain positive), and additionally that we never ADD edges to the graph (ie. checks are written only to people the writer owes money to), all the conditions of a valid reconciliation are met. This method must create a system of debts equivalent to the original.

  This method may be applied until only n-1 edges remain.

  Complexity: We can find a cycle using breadth-first search, which is $O(m + n)$. Finding the minimal cost edge in the cycle is maximally $O(m)$. Likewise, updating the edge costs in the cycle is $O(m)$. So a single iteration of the method is $O(m + n)$. We run this a maximum of $O(m)$ iterations, since we can't run more times than number of edges. So the total algorithm runs in $O(m(m + n))$.

- **Consider the "continuous knapsack problem":**
  You are given a knapsack of capacity k (it can hold up to k grams of stuff) there are powders 1, 2, ... p there is an amount of wi grams of powder i powder i is valued at ci dollars per gram There is no problem mixing powders in the knapsack. Denote by xi (i=1, 2, ..p) the amount of powder that you put in the knapsack. Write as a linear program (with variables xi) the maximization problem that has as solution the values of xi that maximize the value of the knapsack.

  The quantity that we are trying to maximize is the value of the knapsack, which will be the amount of each powder times its unit value. This is our objective function. Our constraints are that the sum of the

powder amounts must be less that or equal to the capacity of the knapsack and we cannot add more of a powder than we have. Obviously, we also cannot add a negative quantity of powder.
Maximize $\sum_{i=1}^{p} c_i x_i$

Subject to constraints:
$x_i \leq w_i$
$\sum_{i=1}^{p} x_i \leq k$
$x_i \geq 0$

- **Extra Credit**