Optimization II
some Homework 3 solutions

Spring 2006

1. Problem 13.30. Most vital arcs.

FindMostCritical$(G = (V, E)), c$

$\triangleright$ Assumes that E is in the form of an adjacency list,
$\triangleright$ For a node v, $A_E(v)$ is all the edges incident to v.
T be the edges of an MST of G $\triangleright$ Use your favorite algorithm
$F \leftarrow E$ $\triangleright$ Working copy of the adj. list
$U \leftarrow V$ $\triangleright$ Working copy of the node list
while $|U| > 1$ $\triangleright$ while U is not a single node
    do
        Set $s \leftarrow$ some leaf of MST
        $E(s) \leftarrow e \in A_T(s)$. $\triangleright$ only one since s is a leaf
        Contract(s, T, F, c)

$t \leftarrow \arg\max_s\{D(s)\}$
return $E(t)$

Figure 1: Procedure FindMostCritical(G)

Contract(s, T, V, E, c)

$\triangleright$ This contracts the node and preserves the non-tree arc costs with the
$\triangleright$ smallest values. Note that s is assumed to be a leaf
Note that $A_T(s) = (s, u)$
for $(s, j) \in A_E(s) \setminus (\{(s, u)\} \cup A_T(u))$
    do
        if $(u, j) \in A_E$
            then $c(s, j) \leftarrow \min\{c(s, j), c(u, j)$
            else $A_E(u) \leftarrow A_E(u) \cup \{(s, j)\}$

        $E \leftarrow E \setminus \{(s, j)\}$

$V \leftarrow V \setminus \{s\}$
$T \leftarrow T \setminus \{(s, u)\}$
return T, V, E

Figure 2: Procedure Contract(s, T, V, E, c)

We first show that FindMostCritical returns the most critical arc.

Lemma 1 *If a most critical arc exists in a graph,* FindMostCritical *will find it.*

Proof: Consider a graph $G = (V, E)$ with a given MST, (V, T). Note that, if a most critical arc exists in this graph, the arc most be in any MST for the graph, as deleting the most critical arc changes the cost of the MST, whereas deleting a non-tree arc will not change the cost of the MST. Then, the cut $[I, J]$ formed by removing an arc $(i, j) \in T$. Note that $I := \{v \in V : \exists$ a path from v to i in $T\}$, and $J = V \setminus I$. In this graph, the Cut Optimality conditions indicate that a new MST can be formed by adding the minimum cost arc that crosses $[I, J]$ to T. The arc corresponding to the tree arc with the largest differential between its cost and the arc with the next smallest cost along the cut is the most vital arc. FindMostCritical does exactly this, by choosing a leaf arc and examining the edges in the cut set formed.

To complete the proof, we note that any minimum spanning tree always has at least two leafs, and a straightforward recursive argument shows that Contract preserves the spanning tree property of T on the contracted graph. Moreover, Contract preserves the minimum cost non-tree arcs for the subsequent node sets in its update. ■

Note that FindMostCritical cannot take advantage of sparseness, since Contract (possibly) creates new neighbors. We formalize this in the following lemma.

Lemma 2 FindMostCritical *has a runtime of* $O(n^2)$.

Proof: Each tree arc is examined once in the outer loop of FindMostCritical. In Contract, each neighbor of s is examined and made a neighbor if its tree neighbor. Since there $n - 1$ possible neighbors of s, and each time a neighbor is examined it incurs $O(1)$ work, the runtime per tree arc is $O(n)$, which results in a total runtime of $O(n^2)$. ■

Lemmas 1 and 2 allow us to conclude that FindMostCritical runs correctly in $O(n^2)$ time. An interesting question is whether an $O(m)$ algorithm exists.

2. Problem 13.34. Balanced Spanning Trees.
Our algorithm forms an MST on the input graph, sorts the edges in ascending cost order, and then, iteratively tries to improve the MST in the following manner. In the sorted order, we will remove an arc from the MST, form the resulting cut with respect to the MST, and add the lowest cost edge. This forms a new spanning tree that is also an MST with respect to a certain subgraph. We compute the difference between smallest and largest cost edge on this tree and repeat. Figure 3 encodes this in a more precise manner.

We can use Kruskal's or Prim's algorithm for finding MST at line 1, both of which run in $O(m+n \log n)$ time. To sort the edges at line 2 requires $O(m \log n)$ time, and computing the minimum and maximum tree cost edge (line 3 in the loop requires $O(n)$ time. If the new tree is an improvement, then we require an additional $O(1) + O(n)$ time for lines 4 and 5. Checking to see if $(i, j) \in T$ at line 6 and forming the cut sets I and J at line 7 requires another $O(n)$. To this point, our operations require $O(n)$ time in the loop and $O(m \log n)$ out of the loop, for a total runtime of $O(nm)$. Our bottleneck occurs on line 8, where we require $O(m)$ to find the minimum cost edge crossing the cut, $[I, J]$. Because of this step, our algorithm runs in $O(m^2)$.

Lemma 3 FindBalancedTree *runs in* $O(m^2)$.

Correctness follows from the following lemma:

Lemma 4 *A most balanced trees on* (V, E, c) *is either an MST on the graph* $(V, E(s), c)$ *where* $d(s)$ *corresponds to the minimum cost edge of the balanced tree, and* $E(s)$ *is defined in line 10 of* FindMostBalanced*, or it can be converted to an MST on* $(V, E(s), c)$ *which is also a most balanced tree on* (V, E, c).

FindBalancedTree(V, E, c)

▷ Finds the most balanced tree in (V, E)
▷ We assume (V, E) is connected
1 Let $T \leftarrow$ MST on (V, E, c).
2 Let d be an array of pointers to the edges sorted in order of c.
$\alpha \leftarrow c(d(m)) - c(d(1)) + 1$ ▷ the best gap
$s \leftarrow 0$ ▷
for $k = 1, \ldots, m$
do
3 Compute: $\beta := \max\{c(e) : e \in T\} - \min\{c(e) : e \in T\}$
if $(\beta < \alpha)$,
then
4 $\alpha \leftarrow \beta$
5 $s \leftarrow \arg\min_t \{t : d(t) \in T, t = 1, \ldots, n\}$
6 **if** $\big((i, j) = d(k) \in T\big)$
then
7 Form $I \leftarrow \{u \in V : \exists$path from u to i in T$\}$, $J \leftarrow V \setminus I$
8 Let $(u, v) = \arg\min_{(a,b)} \{c(a, b) : a \in I, b \in J, (a, b) \notin T$
9 $T \leftarrow (T \setminus \{(i, j)\}) \cup \{(u, v)\}$
10 **return** MST on $(V, c, E(s) \equiv \{e \in E : c(e) \geq c(d(s))\})$

Figure 3: Procedure FindBalancedTree(V, E, c)

Proof: Consider a most balanced tree, T, on (V, E, c), and note that T must also be a most balanced tree on $(V, E(s), c)$. Then suppose it was not an MST on $(V, E(s), c)$, and note, by the equivalence of the path optimality conditions, there must exist nodes u to v such that $c(u, v) < c(e)$ for all e on the path from u to v in T. But, adding $c(u, v)$ to T, and removing the nonessential arc on the resulting cycle creates a new tree which is at least as balanced as T. We can repeat this until we have a tree which satisfies the path optimality conditions for $(V, E(s), c)$. ■

Each time FindMostBalanced computes a new tree at line 9, the new arc added will cause the tree to satisfy the cut optimality conditions for the subgraph $(V, E(d(k))$. Thus, FindMostBalanced finds the most balanced tree amongst the MSTs for the graphs from the set: $\{(V, E(d(k)) : k = 1, \ldots, m\}$. But then Lemma 4 implies the correctness of the algorithm. Together with Lemma 3, this implies that FindMostBalanced runs correctly in $O(m^2)$ time.