15-451 Assignment 05

Karan Sikka
ksikka@cmu.edu
Collaborated with Dave Cummings and Sandeep Rao.
Recitation: A
October 10, 2014

1: Cell Towers

Profit = Actual Revenue - Cost
Profit = Total Possible Revenue - Lost Revenue - Cost
LR + Cost = TPR - Profit

Since the total possible revenue is some constant, minimizing the LHS of the above equation maximizes the profit.
So finding the min (lost revenue + cost) is the right thing to do. Now lets see how to do it.

In part (a) we will construct a graph.
In part (b) we will show that the finding the min cut correctly finds the cell towers to build which min (lost revenue + cost)
In part (c) we will give an algorithm to find the min cut.
In part (c) we will show that it's efficient.

(a) Construct a bipartite graph (nodes are partitioned into A and B with no edges within A or B) as follows:

There is a node in A for every revenue-generating pair of cell towers.
Add a source node.
Create edges from the source node to every node in A with capacity of the revenue generated by the pair of cell towers represented by the node in A.

For each node in A, for each cell tower in the pair represented by that node, connect that node to the cell tower in B.
Give that edge an infinite capacity.
There is a node in B for every cell tower.
Add a target node.
Create edges from every node in B to the target with capacity of the cost of building the tower represented by the node in B.

(b) Notice that the min cut will never contain an edge going from A to B, because those edges have infinite capacity.
Therefore the node representing a pair of cell towers in A will be in the same vertex set as the individual cell towers in B after the cut is performed.

If the min cut contains an edge from B to target, we say that we built that cell tower of the node in B, and incurred a financial cost of the amount of that capacity.
If the min cut contains an edge from source to A, we say that at least one of the cell towers in pair of the node in A will not be built, and we lost revenue of the amount of that capacity.

Min-cut will minimize the cost + the lost revenue

The edges of the min cut from B to target will tell you which cell towers to build.

(c) You can perform Edmund-Karp #1 until your last residual graph when the algorithm terminates. On the residual graph, find all vertices reachable from the source. This is one set in the min cut, the other set has the nodes not in the first set.
This is correct because the algorithm pushes max flow until some edges have residual capacity 0 and they disappear, disconnecting the graph. This max flow value is exactly equal to the min cut value. Therefore the vertices not reachable must have
been separated by the edges of the min cut.

Find the edges between the sets, look for edges where one node is the target, and the other node in the edge is a cell tower you should build.

(d)

The number of nodes and edges in the graph is no more than polynomial in the number of cell towers and revenue pairs due to the way we constructed the graph.
The Edmund-Karp algorithm is polynomial in nodes and edges.
Therefore this algorithm runs in polynomial time and is "efficient".

2: Eliminating Negative Edges

(a) Original graph had no negative cycles. We introduce a node s, create an edge from s to every other node, and give edge weight 0.
A new negative cycle would have to include s. However that can't be because there are no edges going into s. Therefore there are no negative cycles in the new graph.

(b)

Case the shortest path to v goes through u.
$\Phi(v) = \texttt{shortest\_path\_from\_u\_to\_v} + \Phi(u)$
and we know that $\texttt{shortest\_path\_from\_u\_to\_v} \leq \texttt{len}(u, v)$
$\implies \Phi(v) \leq len(u, v) + \Phi(u) \implies \Phi(v) - \Phi(u) \leq len(u, v)$.

Case the shortest path to v does not go through u.
Then the shortest path from s to v is even shorter than the shortest path that goes through u, so $\Phi(v)$ is smaller than was shown in the previous case, so the original claim holds.

(c) Consider a path of nodes $v_1, v_2, ..., v_k$. When you sum the new edge lengths, you see all of the Φ terms cancel out except
you are left with $\Phi(v_1) - \Phi(v_k)$. So the length of a path in the new graph is $\texttt{old\_path\_len} + \texttt{Φ}(\texttt{v}_1) - \texttt{Φ}(\texttt{v}_k)$

For each node, run djikstra's single source shortest path algorithm.
We claim that the resulting shortest paths will be the true shortest paths in the original graph.
This is because the shortest path from some u to some v is going to be the path among all paths from u to v minimizing $\texttt{old\_path\_len}(\texttt{u}, \texttt{v}) + \texttt{Φ}(\texttt{u}) - \texttt{Φ}(\texttt{v})$ where $\Phi(u) - \Phi(v)$ is a constant quantity for a given u, v.
The real length of the path from u to v is $\texttt{new\_path\_len}(\texttt{u}, \texttt{v}) - \texttt{Φ}(\texttt{u}) + \texttt{Φ}(\texttt{v})$
The cost of running djikstra's n times using the fibonacci heaps version is $O(nm + n^2 lg(n)$

3: Color Me Red, Color Me Blue

For convenience, we will call a $(k, n-k)$ partition which minimizes split edges a "minimizing k-partition".

We will consider a function kPart as follows:

Given a node n, value $i(1 \leq i \leq k)$ and bit b,
The function returns NONE, or SOME (A, B, S).

`kPart`(n, k, b) = `NONE` signifies that no k-partition of vertices in the subtree rooted at n exists, which happens when $(k > |n|)$.
`kPart`(n, k, b) = `SOME`(A, B, S) signifies that A, B is some minimizing k-partition the nodes of the subtree rooted at n such that $n \in A \iff b = 1$, S is the set of split edges

We create a memo table for the computation which we assume is available to the function.
The table can be accessed by $T[n][k][b]$ where n is a node, k is an int from 0 to k, b is a bit.
We initialize the table with NULLs, and compute the values in the following recursive function.
Note that the table has $n * (k+1) * 2$ cells.

The procedure to compute kPart(r,k,b) is as follows:

(Check if value has already been computed.)
If $T[r][k][i] \neq$ NULL,
    return $T[r][k][i]$
(If k=0, A is empty, B is rest of nodes.)
If $k = 0$,
    $T[r][k][i] = (\{\}, \texttt{all\_nodes}(b), \{\})$
(If k >n, no set A can be formed...)
If $k > n$,
    $T[r][k][i] =$ `DNE`
If both `r->left` and `r->right` are NULL,
    If $k = 1$
        $T[r][k][i] = (\{r\}, \{\}, \{\})$
    Else
        k >n and this case would have been caught earlier
If only `r->left` is NULL,
    (There are four cases.)
    (We will compute the number of split edges in each case,)
    (take the case with minimum split edges)
    (return the partitioning for that case.)
    $T[r][k][i]$ = The (A, B, S) with the minimum $|S|$ over the following answers computed for all possible colorings of r and `r->right` that might yield a minimizing k-partition:
        $r \in A$, `r->right` $\in A \implies$
            $A_r, B_r, S_r =$ `kPart(r->right` $, k-1, 1)$
            answer is $(\{r\} \cup A_r, B_r, S_r)$
        $r \in A$, `r->right` $\in B \implies$
            $A_r, B_r, S_r =$ `kPark(r->right` $, k-1, 0)$
            answer is $(\{r\} \cup A_r, B_r, S_r \cup \{(r,$ `r->right` $)\})$
        $r \in B$, `r->right` $\in A \implies$
            $A_r, B_r, S_r =$ `kPark(r->right` $, k, 1)$
            answer is $(A_r, \{r\} \cup B_r, S_r \cup \{(r,$ `r->right` $)\})$

$r \in B, \texttt{r->right} \in B \implies$
$A_r, B_r, S_r = \texttt{kPark}(\texttt{r->right}, k, 0)$
answer is $(A_r, \{r\} \cup B_r, S_r)$

If only `r->right` is NULL,

Logic is symmetric to above case.

If `r->left` and `r->right` are not NULL,

$T[r][k][i]$ = The (A, B, S) with the minimum $|S|$ over the following answers computed for all possible colorings of r, `r->right` , and `r->left` that might yield a minimizing k-partition:

Left and right subtrees can be partitioned in at most k+1 ways such that the sum of their A sets should be $k-1$ if r in A or k otherwise.
We must minimize the number of split edges accross all variations of A-set-size.

$r \in A, \texttt{r->left} \in A, \texttt{r->right} \in A$
$\texttt{minimize}|S|\texttt{over\_all\_integral} i, j \geq 0 s t i + j = k - 1$

$A_l, B_l, S_l = \texttt{kPart}(\texttt{r->left}, i, 1)$
$A_r, B_r, S_r = \texttt{kPart}(\texttt{r->right}, j, 1)$
$\texttt{over\_all\_integral}(A_l \cup A_r \cup \{r\}, B_l \cup B_r, S_l \cup S_r)$

$r \in A, \texttt{r->left} \in A, \texttt{r->right} \in B$
$\texttt{minimize}|S|\texttt{over\_all\_integral} i, j \geq 0 s t i + j = k - 1$
$\texttt{kPart}(\texttt{r->left}, i, 1) + 1 + \texttt{kPart}(\texttt{r->right}, j, 0)$
$\texttt{over\_all\_integral}(A_l \cup A_r \cup \{r\}, B_l \cup B_r, S_l \cup S_r \cup \{(r, \texttt{r->right})\})$

$r \in A, \texttt{r->left} \in B, \texttt{r->right} \in A$
$\texttt{minimize}|S|\texttt{over\_all\_integral} i, j \geq 0 s t i + j = k - 1$
$1 + \texttt{kPart}(\texttt{r->left}, i, 0) + \texttt{kPart}(\texttt{r->right}, j, 1)$
$\texttt{over\_all\_integral}(A_l \cup A_r \cup \{r\}, B_l \cup B_r, S_l \cup S_r \cup \{(r, \texttt{r->left})\})$

$r \in A, \texttt{r->left} \in B, \texttt{r->right} \in B$
$\texttt{minimize}|S|\texttt{over\_all\_integral} i, j \geq 0 s t i + j = k - 1$
$1 + \texttt{kPart}(\texttt{r->left}, i, 0) + 1 + \texttt{kPart}(\texttt{r->right}, j, 0)$
$\texttt{over\_all\_integral}(A_l \cup A_r \cup \{r\}, B_l \cup B_r, S_l \cup S_r \cup \{(r, \texttt{r->left}), (r, \texttt{r->right})\})$

$r \in B, \texttt{r->left} \in A, \texttt{r->right} \in A$
$\texttt{minimize}|S|\texttt{over\_all\_integral} i, j \geq 0 s t i + j = k$
$1 + \texttt{kPart}(\texttt{r->left}, i, 1) + 1 + \texttt{kPart}(\texttt{r->right}, j, 1)$
$\texttt{over\_all\_integral}(A_l \cup A_r, B_l \cup B_r \cup \{r\}, S_l \cup S_r \cup \{(r, \texttt{r->left}), (r, \texttt{r->right})\})$

$r \in B, \texttt{r->left} \in A, \texttt{r->right} \in B$
$\texttt{minimize}|S|\texttt{over\_all\_integral} i, j \geq 0 s t i + j = k$
$1 + \texttt{kPart}(\texttt{r->left}, i, 1) + \texttt{kPart}(\texttt{r->right}, j, 0)$
$\texttt{over\_all\_integral}(A_l \cup A_r, B_l \cup B_r \cup \{r\}, S_l \cup S_r \cup \{(r, \texttt{r->left})\})$

$r \in B, \texttt{r->left} \in B, \texttt{r->right} \in A$
$\texttt{minimize}|S|\texttt{over\_all\_integral} i, j \geq 0 s t i + j = k$
$\texttt{kPart}(\texttt{r->left}, i, 0) + 1 + \texttt{kPart}(\texttt{r->right}, j, 1)$
$\texttt{over\_all\_integral}(A_l \cup A_r, B_l \cup B_r \cup \{r\}, S_l \cup S_r \cup \{(r, \texttt{r->right})\})$

$r \in B, \texttt{r->left} \in B, \texttt{r->right} \in B$
$\texttt{minimize}|S|\texttt{over\_all\_integral} i, j \geq 0 s t i + j = k$
$\texttt{kPart}(\texttt{r->left}, i, 0) + \texttt{kPart}(\texttt{r->right}, j, 0)$
$\texttt{over\_all\_integral}(A_l \cup A_r, B_l \cup B_r \cup \{r\}, S_l \cup S_r)$

return $T[r][k][i]$

Each function call costs $O(1)$ if the value in the table is computed, else it makes recursive calls, but these are also $O(1)$ if computed already. The base case takes $O(1)tocompute$. We see that each function call we makes at most $8(k+1)$ recursive function calls, we do this a total of n times. That's at most O(nk) function calls total accross the entire algorithm.