

CS 381 HW 7

Ankur Dhoot

November 12, 2016

Q1

(a) Finding a min-cut in G is simple. Suppose we run Edmonds-Karp on G . Look at the final residual graph, G_f , which doesn't contain any augmenting paths. Let S be the set of vertices reachable from s and let $T = V - S$. Clearly $s \in S$ and $t \in T$ and $S \cup T = V$ meaning (S, T) is a valid cut. As is shown in Theorem 26.6 in CLRS, the cut is a min-cut since the value of the flow equals the capacity of the cut. We can efficiently find the set S by running BFS or DFS in G_f from s and then the min-cut is simple $(S, V-S)$. The time taken to run Edmonds-Karp is $O(VE^2)$ and the search to find the set S runs in time $O(V + E)$. Thus, the run-time is dominated by Edmonds-Karp and is $O(VE^2)$.

Algorithm 1 Find a min-cut in G

```
1: function FINDMINCUT( $G, s, t$ )
2:   Run Ford-Fulkerson on  $G$ 
3:   BFS in  $G_f$  from  $s$  to get  $(S, V - S)$ 
4:   return  $(S, V-S)$ 
5: end function
```

(b) I will prove the following claim which will result in an immediate algorithm for determining whether G has a unique min-cut.

Claim: Run Ford-Fulkerson on G and let G_f be the resulting residual graph. Let S be the set of vertices reachable from s in G_f and let T be the set of vertices that have a path to t in G_f . (Note that $S \cup T = V$ is not necessarily true, which is essentially the crux of the proof). Then $(S, V-S) \neq (V-T, T)$ if and only if there exists more than one min-cut in G .

Proof: \implies Suppose $(S, V-S) \neq (V-T, T)$. We know that $(S, V-S)$ defines a min-cut. It's also clear that $(V-T, T)$ defines a min-cut (Exact same proof as given in Theorem 26.6 in CLRS easily shows this). Thus, we have more than one min-cut in G .

\Leftarrow Suppose there exists more than one min-cut in G . In G_f , we know there exists no residual edges from S to $V-S$, and we know $(S, V-S)$ is a min-cut. Let (X,Y) be another min-cut of G . No vertices of Y can be in S . Why? A min-cut partitions the vertices such that there are no residual edges from X to Y in G_f meaning there exists no path from s to any vertex in Y in G_f . Thus, by the definition of S , no vertex in Y is a vertex in S . $\Rightarrow Y \subset V - S$. We know $Y \neq V - S$ since $(X,Y) \neq (S, V-S)$. We also know that $T \subset V - S$, since no vertex in T can be in S , else we'd have an augmenting path in G_f . Now we need to show that $T \neq V - S$. Suppose, for the sake of contradiction that $T = V - S$. \Rightarrow all vertices in $V - S$ have a path to t . But $Y \subset V - S$ and $Y \neq V - S \Rightarrow (X,Y)$ splits $V - S$ such that not all vertices in $V - S$ can have a path to t . Why? Suppose all vertices in $V - S$ have a path to $t \Rightarrow$ some vertex $x \in X$ has a path to t . At some point, the path from x to t must go from a vertex in X to a vertex in Y (since $t \in Y$). But this contradicts the fact that (X, Y) is a min-cut meaning there are no residual edges from X to Y in G_f . Thus, not all vertices in $V - S$ have a path to $t \Rightarrow T \neq V - S$.

Having proved both directions of the statement, the proof is complete.

The procedure follows immediately :

Algorithm 2 Find and print two distinct min-cuts if they exist

```

1: function PRINTDISTINCTMINCUTS( $G, s, t$ )
2:   Run Ford-Fulkerson on  $G$ 
3:   BFS in  $G_f$  from  $s$  to get  $(S, V - S)$ 
4:   Reverse  $G_f$  to get  $G_f^R$ 
5:   BFS in  $G_f^R$  from  $t$  to get  $(V - T, T)$ 
6:   if  $(S, V-S) == (V-T, T)$  then
7:     print("Min-cut is unique")
8:   else
9:     print( $S, V-S$ )
10:    print( $V-T, T$ )
11:   end if
12: end function

```

The run-time of Ford-Fulkerson using Edmonds-Karp is $O(VE^2)$. Lines 3 and 5 runs in $O(V+E)$ and line 4 can be done in $O(V+E)$ as well. Thus, the run-time is dominated by Edmonds-Karp and is $O(VE^2)$.

Q2

We'll reduce the problem to finding the median in a set of numbers which we know can be done in linear time.

Given the line $y = mx + b$, let $d_i = (mx_i + b) - y_i$, which represents the (signed) vertical distance between the i th point and the line. Find the median of these d_i values, and call it d^* and its associated point (x^*, y^*) . Let the new line that is parallel to the original line, but divides the points into equal-sized subsets pass through (x^*, y^*) . Using middle-school rules for lines, we get that the new line has equation $y = mx + (y^* - mx^*)$. Since the new line is simply a vertical shift of the original, it's clear that the relative ordering of the d_i values will remain the same when computed using the new line. Thus, the median, d^* and its associated point (x^*, y^*) , remain the same \Rightarrow half the points have d values $\geq d^*$ and half the points have d values $\leq d^*$. The line passes through $(x^*, y^*) \Rightarrow$ half the points lie on or above the line and half the points lie on or below the line.

The algorithm follows :

Algorithm 3 Find parallel line that divides points into equal-sized subsets

```
1: function FINDLINE(P)
2:   compute the  $d$  values for all points in  $P$ 
3:   Let  $d^*$  be the median of the  $d$  values with corresponding point  $(x^*, y^*)$ 
4:   Compute the parallel line through  $(x^*, y^*)$ 
5:   return this new parallel line
6: end function
```

Line 2 takes $O(n)$ time. Line 3 can be done in $O(n)$ time. Line 4 takes constant time. Thus, the algorithm runs in $O(n)$ time.