DESIGN AND ANALYSIS OF ALGORITHMS
Homework 6

Aman Choudhary
MTech Coursework, CSA 2020
Sr No: 17920

November 27, 2020

1 Problem 1

1.1 Algorithm

Ford-Fulkerson Algorithm

initialize $f_e = 0$ for all $e \in E$
repeat
  search for an s-t path P in the current residual graph G_f such that every edge of P has positive residual capacity
  `// takes` $O(|E|)$ `time using BFS or DFS`
  if no such path **then**
    halt with current flow $\{f_e\}_{e \in E}$
  else
    let $\Delta = \min_{e \in P}$ (e's residual capacity in G_f)
    `// augment the flow` f `using the path` P
    for all edges e of G whose corresponding forward edge is in P **do**
      increase f_e by Δ
    for all edges e of G whose corresponding reverse edge is in P **do**
      decrease f_e by Δ

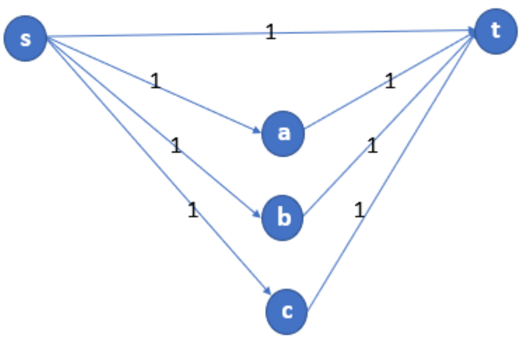
1.2 Time Complexity

(i) We run the loop while there exists an augmenting path. In every iteration, we can augment the graph with exactly 1 unit of flow in this **unit capacity** case. This is because, if BFS find an $s - t$ path, then $\Delta > 0$. Moreover, $\Delta \leq 1$, because the residual capacity for any edge cannot exceed the actual capacity for that edge. The above two conditions imply that $\Delta = 1$ for any such path.

(ii) The source s can have an outgoing edge to every other vertex, and each such edge can have a maximum capacity of 1. Hence, the maximum flow possible in the unit capacity case, is $|V| - 1$. Since, we can augment exactly single unit per flow, the loop would iterate for at most $|V| - 1$ times in this case. The loop terminates after reaching the maximum possible flow.

(iii) During each iteration, we perform BFS and if we find an (s, t) path, we augment the flow using the path. Both BFS and the augmentation take $O(|E|)$ time.

(iv) Hence, the time complexity of the above algorithm is $O(|V| - 1) * |E|) = O(|V||E|)$.



The **maximum flow** possible in the **unit capacity case**, is **V-1**, because the source **s** could have at most V-1 **outgoing edges** each with a maximum capacity of 1. In this case, V = 5 and maximum flow is 4.

2 Problem 2

2.1 Notation

(i) $G = (V, E)$ is a directed graph, with source $s \in V$, sink $t \in V$, and non-negative edge capacities c_e.

(ii) An $(s,t) - cut$ of a graph $G = (V, E)$ is a partition of V into sets A, B with $s \in A$ and $t \notin B$.

(iii) The capacity of an $(s,t) - cut(A, B)$ is defined as $\sum\limits_{e \,\in\, \delta^+_G(A)} c_e$ where c_e denotes the set of edges sticking out of A in G.

2.2 Algorithm

(i) With s as source and t as sink, run **Edmond-Karp** algorithm on graph G. Let G_r be the residual network when the Edmond-Karp algorithm halts.

(ii) With s as source, we run **BFS** on the residual network G_r until we get stuck. Let, A be the set of vertices we get stuck at. We note that $(A, V - A)$ is an $(s,t) - cut$. Certainly $s \in A$, since s can reach itself in G_r. By assumption, G_r has no $s - t$ path, so $t \notin A$.

(iii) Process graph $G = (V, E)$ to produce a new graph $G' = (V, E')$ such that the direction of each edge in G is **reversed** in G'.

(iv) With t as source and s as sink, run **Edmond-Karp** algorithm on graph G'. Let G'_r be the residual network when the Edmond-Karp algorithm halts.

(v) With t as source, we run **BFS** on the residual network G'_r until we get stuck. Let, B be the set of vertices we get stuck at. We note that $(B, V - B)$ is a $(t,s) - cut$. Certainly $t \in B$, since t can reach itself in G'_r. By assumption, G'_r has no $t - s$ path, so $s \notin B$.

(vi) Now, there are two cases:

   (a) **The cuts** $(A, V - A)$ **and** $(V - B, B)$ **are identical**. That is, $A = V - B$, and $V - A = B$. In this case, there exists a **unique minimum** $(s,t) - cut$ in G, which is precisely $(A, V - A)$ (which is same as $(V - B, B)$).

   (b) **The cuts** $(A, V - A)$ **and** $(V - B, B)$ **are not-identical**. That is, $A \neq V - B$ (and $V - A \neq B$). In this case, the **minimum** $(s,t) - cut$ in G is **not unique**.

2.3 Proof of Correctness

CLAIM 1: The **maximum flow** in graph G' is equal to the **maximum flow** in graph G. This can be proofed by contradiction.

(a) Let us assume, **max-flow** $(G') >$ **max-flow** (G). If this is so, then we can simply reverse the edges of G' to get G, while keeping the flows in G, same as that of flow in G'. Hence, we will obtain a greater flow for G, than what was produced by Edmonds-Karp algorithm. This is not possible, and we reach a contradiction.

(b) Let us assume, **max-flow** $(G') <$ **max-flow** (G). If this is so, then we can simply reverse the edges of G to get G', while keeping the flows in G', same as that of flow in G. Hence, we will obtain a greater flow for G', than what was produced by Edmonds-Karp algorithm. This is not possible, and we reach a contradiction.

As a result, **max-flow** (G) **=** **max-flow** (G')

CLAIM 2 : The $(s,t) - cut$, $(A, V - A)$ generated using BFS in step (ii) is a **minimum cut** of graph $G = (V, E)$. This follows from the proof of **3rd part of Theorem 2.2** in lecture notes. Similarly, the $(t,s) - cut$, $(B, V - B)$ generated using BFS in step (v) is a **minimum cut** of graph $G' = (V, E')$.

CLAIM 3: For every minimum $(t,s) - cut$, $(B, V - B)$ in G', there exists a corresponding minimum $(s,t) - cut$, $(V - B, B)$ in G. The reverse is also true, that for every minimum $(s,t) - cut$, $(A, V - A)$ in G, there exists a corresponding minimum $(t,s) - cut$, $(V - A, A)$ in G'.

For proving the first part, let us consider the $(t,s) - cut$, $(B, V - B)$ in G',

$$\textbf{Capacity of } (t,s)-cut\textbf{, } (B, V-B) \textbf{ in } G' = \sum_{e \,\in\, \delta^+_{G'}(B)} c_e = \sum_{e \,\in\, \delta^-_{G}(B)} c_e = \sum_{e \,\in\, \delta^+_{G}(V-B)} c_e =$$
$$\textbf{Capacity of } (s,t)-cut\textbf{, } (V-B, B) \textbf{ in } G$$

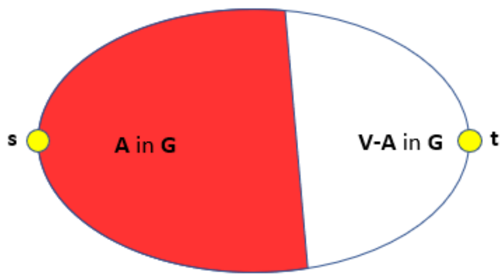
Also,
Capacity of $(s,t)-cut$**,** $(V-B, B)$ **in** G = Capacity of $(t,s)-cut$, $(B, V-B)$ in G' (using **CLAIM 3**)
Capacity of $(t,s)-cut$, $(B, V-B)$ in G' = Max-flow in G' (using **Max-Flow/Min-Cut Theorem**)
Max-flow in G' = **Max-flow in** G (using **CLAIM 1**)

Thus, we have established the existence of a **minimum** $(s,t)-cut$, $(V-B, B)$ in G corresponding to the minimum $(t,s)-cut$, $(B, V-B)$ in G'. The reverse implication can be proved using a similar argument.

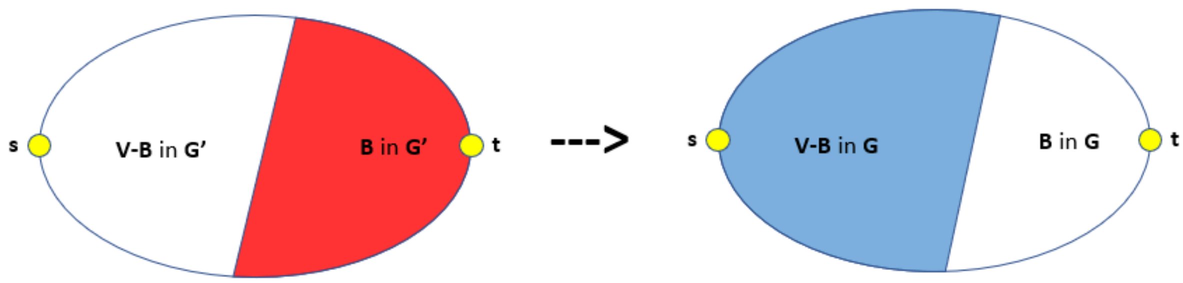
In this figure, we show how our algorithm finds two minimum $(s,t)-cuts$ for G.

(i) $(A, V-A)$ **[using CLAIM 2, discovered in step (ii)]**

(ii) $(V-B, B)$ **[using CLAIM 3, discovered in step (v)]**



s-t cut **(A, V-A)** in **G** discovered in Step (ii) of Algorithm



t-s cut **(B , V-B)** in **G'** discovered in Step (v) of Algorithm

Implies existence of s-t cut **(V-B , B)** in **G**

Now consider the two cases:

(a) **The graph** G **indeed has a unique minimum** $(s,t)-cut$**,** (A, B). Then, in step (ii) we will discover the minimum $(s,t)-cut$, (A, B) for G, while in step (v) we would discover the minimum $(t,s)-cut$, (B, A) for G'. In this case, the $(s,t)-cut$, (A, B) **implied** by the $(t,s)-cut$, (B, A) will be identical to the cut discovered in step (ii).
Let us assume the other way round, i.e. we find in step (vi), that the $(s,t)-cut$, $(A, V-A)$ and $(V-B, B)$ are identical. This implies that the cut $(A, V-A)$ is unique. This is because, the cut $(A, V-A)$ is the closest $(s,t)-cut$ to source s in G. The BFS would get stuck at this cut, and would not explore any further. Moreover, the cut $(B, V-B)$ is the closest $(t,s)-cut$ to source t in G'. If there existed any other $(s,t)-cut$, $(X, V-X)$ after $(A, V-A)$ in G, then there would exist a corresponding $(t,s)-cut$, $(V-X, X)$ in G', and it would become the closest cut to source t in G'. In such a case, the algorithm would have reported the $(t,s)-cut$, $(V-X, X)$, instead of $(B, V-B)$ in step (v).

(b) **The graph** G **has more than one minimum** $(s,t)-cut$. Let us assume the cut which is closest to source s in G is $(X_1, V-X_1)$, and the cut which is farthest from s, is $(X_2, V-X_2)$. This $(s,t)-cut$, $(X_2, V-X_2)$ would thus be closest to sink t. In step (ii), we will discover the $(s,t)-cut$, $(X_1, V-X_1)$. The BFS would get stuck at this cut, and would not explore any cuts which lie farther from s. Similarly, in step (v), we will discover the $(t,s)-cut$, $(V-X_2, X_2)$. The BFS would get stuck at this cut, and would not explore any cuts which lie farther from t. In this case, the $(s,t)-cut$, $(X_2, V-X_2)$ **implied** by the $(t,s)-cut$,

$(V - X_2, X_2)$ will not be identical to the $(s,t) - cut$, $(X_1, V - X_1)$ discovered in step (ii).
Let us assume the other way round, i.e. we find in step (vi), that the $(s,t) - cut$, $(A, V - A)$ and $(V - B, B)$ are not-identical. Then it is trivially true, that the graph has more than one minimum $(s,t) - cut$.

Therefore, checking for equality of cuts (step (vi) of algorithm), provides the condition for uniqueness.

2.4 Time Complexity

Let us assume, $m = |E|$, and $n = |V|$.

1. In, step (i) and (iv) we run **Edmonds-Karp** algorithm which runs in $O(m^2n)$ time, no matter how big the edge capacities are.
2. In, step (ii) and (v) we run **BFS** algorithm, which takes $O(m)$ time.
3. In step (iii), we use G to produce G'. This can also be done in $O(m)$ time, by a simple processing of all the edges.

Here, the dominant step takes $O(m^2n)$ time. Now, m typically varies between $\approx n$ (the sparse case) and $\approx n^2$ (the dense case), so the running time is between n^3 and n^5. Hence,

Total Time Complexity = $O(n^5)$
