ADVANCE ALGORITHM ANALYSIS Lecture 9 Oct 3, 2017

Maximum Flow Networks

Instructor: *Dr. Mudassir Shabbir* Scribe: Mussa

1 Problem: Shortest path in undirected and unweighted graph

Consider an undirected graph G with V vertices and E edges, as shown in Figure 1. The edges in the graph has either no weight or all edges have equal weight. The shortest distance to any node can be computed by BFS.

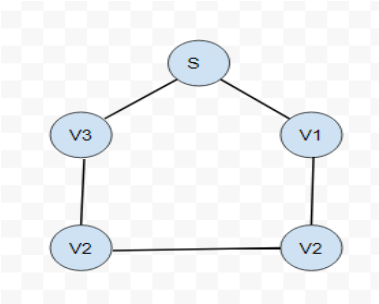


Figure 1: An undirected graph with no weights on edges.

1.1 Question 1

What will be the shortest path from vertex S to V4 ?

Solution: Shortest path can be from vertex S to any vertex can be found by BFS. In case of unweighted graph resultant tree will be **Minimum spanning tree.**

Explanation: By apply Breadth First Search(BFS) we will get the tree as shown in Figure 2. The shortest path from S to V4 is, S $\rightarrow$ V3 $\rightarrow$ V4.

At any time in the BFS queue the following invariant holds:

- Being unweighted adjacency is always shortest path to any adjacent node.
- Therefore, any unvisited non-adjacent node adjacent to adjacent nodes is on the shortest path discovered like this.

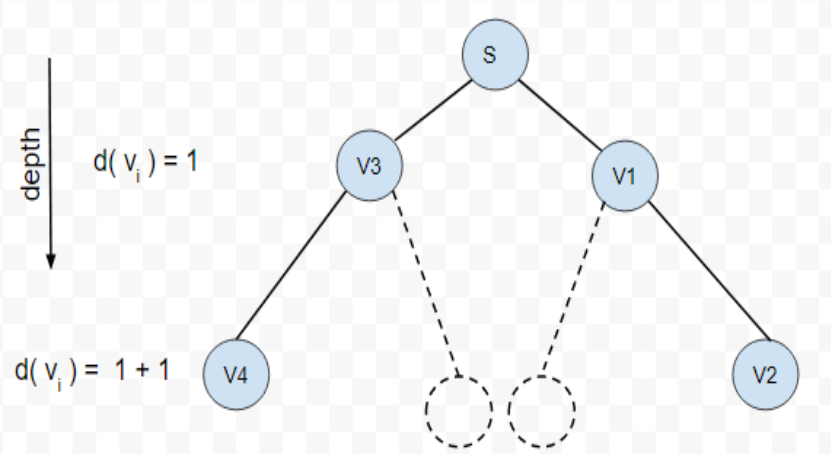


Figure 2: BFS on graph in Figure 1.

- Now, in BSF all we do is, we insert first node in a queue then we keep removing nodes from rear and keep inserting their adjacent nodes in the same queue if they are not already visited. This restricts visiting a node adjacent node before all adjacent ones. So, this is exactly the procedure which uses points 1 and 2 above. Hence, nodes discovered by BFS are through shortest path only.

2 Problem 2: Maximum Flow Network Problem

Flow network problem can be defined as, In the maximum-flow problem, we are given a flow network G(V,E) with source s and sink t, and we wish to find a flow of maximum value.

2.1 Explanation:

A flow network can model the trucking problem shown in Figure 3. In the given flow network G(V,E) for the Lucky Puck Company's trucking problem. The Vancouver factory is the source s, and the Winnipeg warehouse is the sink t. Company ships pucks through intermediate cities, but only c(u,v) crates per day can go from city u to city v. Each edge is labeled with its capacity. A flow f in G with value —f— = 19. Each edge (u,v) is labeled by f(u,v)/ c(u,v). The slash notation merely separates the flow and capacity; it does not indicate division. Our goal is to maximize the flow from node s to t. In this case maximize the flow of crates from Vancouver to Winnipeg.

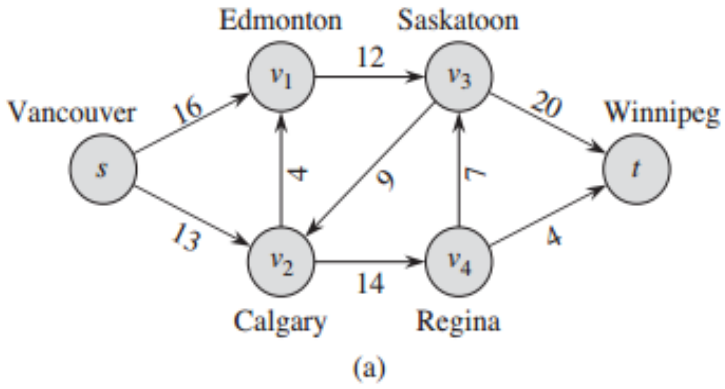


Figure 3: Example of Flow

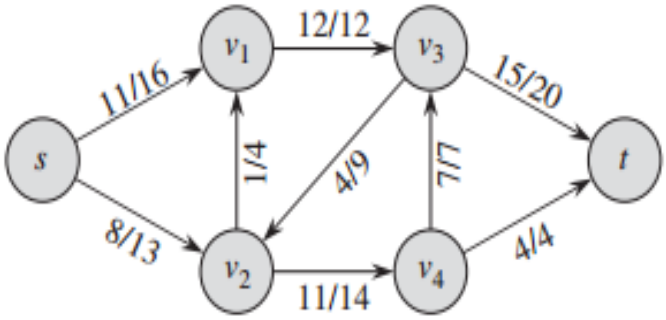


Figure 4: Flow from source

2.2 Conditions of flow

Capacity constraint : For all (u,v) $\in V$, $0 \leq f(u,v) \leq c(u,v)$
Flow conservation : For all u $\in V$ - {s , t} , $\sum_{v \in V} f(u,v) = \sum_{v \in V} f(v,u)$

2.3 Solutions

One solution can be find all possible path from s to t and return the one with maximum flow. But this solution will give us result but it will be in exponential time. The two better algorithms for finding maximum flow from node s to t are.

- Ford-Fulkerson method
- Edmonds-Karp algorithm

3 Ford-Fulkerson method

FORD-FULKERSON-METHOD(G(s,t))

1. initialize flow f to 0
2. while there exists an augmenting path p in the residual network G_f
3. augment flow f along
4. return f

3.1 Residual Network

Intuitively, given a flow network G and a flow f , the residual network G_f consists of edges with capacities that represent how we can change the flow on edges of G.An edge of the flow network can admit an amount of additional flow equal to the edge's capacity minus the flow on that edge. If that value is positive, we place that edge into G_F with a "residual capacity" of $c_f(u, v)$ = c(u,v) - f(u,v). The only edges of G that are in G_f are those that can admit more flow; those edges (u,v) whose flow equals their capacity have $c_f(u, v) = 0$, and they are not in G_f .

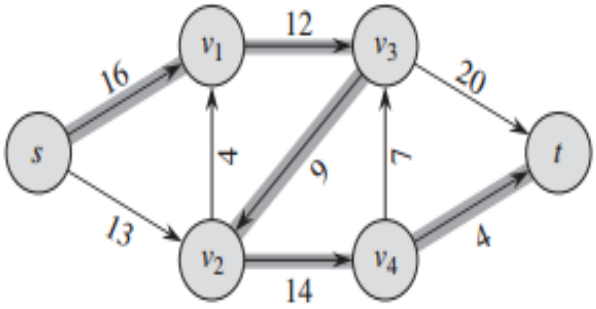
3.2 augmented path

Given a flow network G(V , E) and a flow f , an augmenting path p is a simple path from s to t in the residual network G_f .

3.3 Explanation

Repeatedly run DFS from source node s to sin t until there's no path exists in residual network from sorce s to sink t. The Ford-Fulkerson method iteratively increases the value of the flow. We start with f(u)u,v) = 0 for all u,v ∈ V , giving an initial flow of value 0. At each iteration, we increase the flow value in G by finding an "augmenting path" in an associated "residual network" G_f . Once we know the edges of an augmenting path in G_f , we can easily identify specific edges in G for which we can change the flow so that we increase the value of the flow. Although each iteration of the Ford-Fulkerson method increases the value of the flow, we shall see that the flow on any particular edge of G may increase or decrease; decreasing the flow on some edges may be necessary in order to enable an algorithm to send more flow from the source to the sink. We repeatedly augment the flow until the residual network has no more augmenting paths. The max-flow min-cut theorem will show that upon termination, this process yields a maximum flow.

3.4 Example of Fork-Fulkerson :



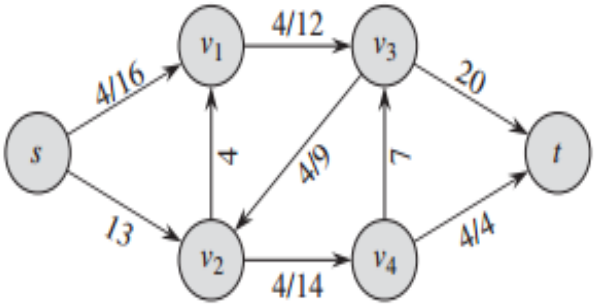
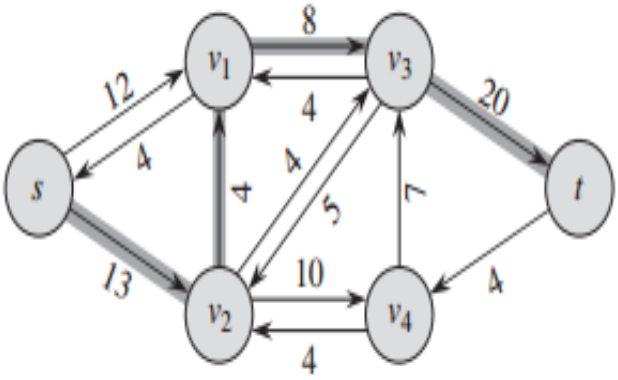


Figure 5:



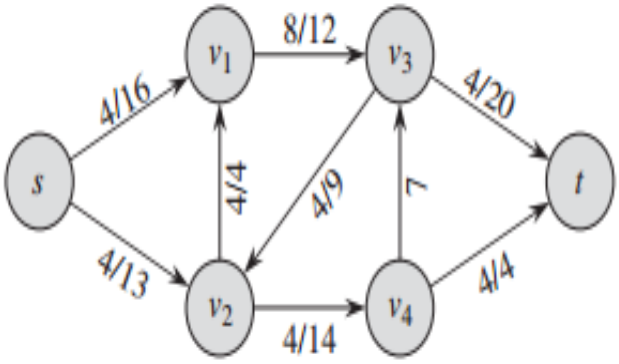


Figure 6:

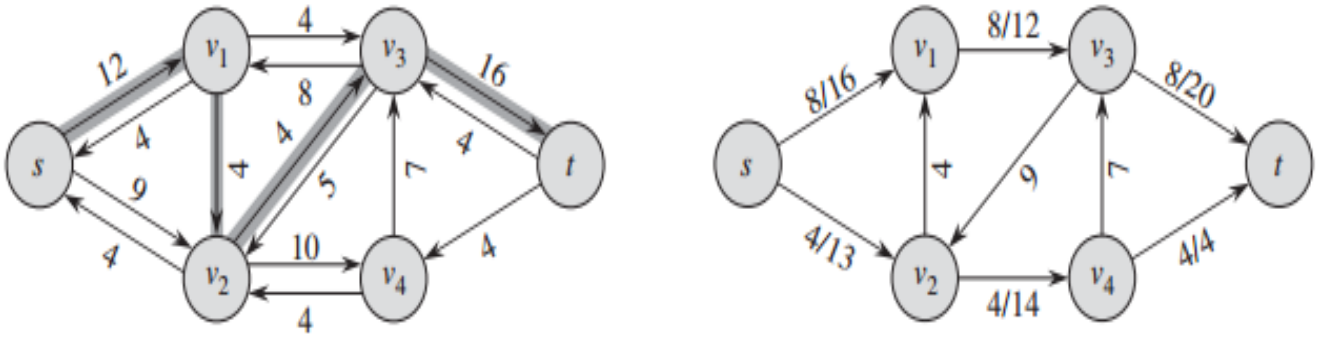


Figure 7:

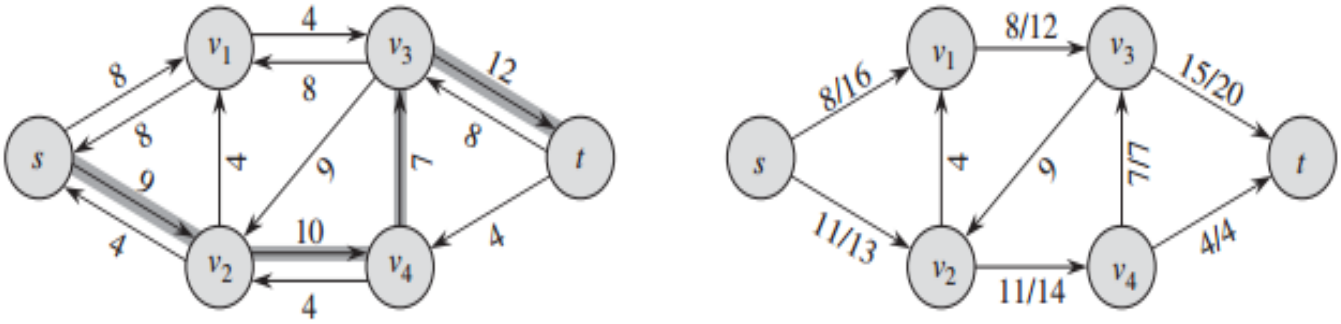


Figure 8:

3.5 Running time of Fork-fulkerson algorithm

In practice, the maximum-flow problem often arises with integral capacities. If the capacities are rational numbers, we can apply an appropriate scaling transformation to make them all integral. If f^* denotes a maximum flow in the transformed network, then a straightforward implementation of FORD-FULKERSON executes the while loop of lines 3–8 at most $—f^*—$ times, since the flow value increases by at least one unit in each iteration.
Running time of fork-fulkerson algorithm will be O(((V+E).f). where f is the maximum flow. Example in figure 11. This figure illustrates worst case of fork-fulkerson algorithm.If the path in odd iterations is chosen as s ->u ->v ->t and in case of even numbers the path chosen is s ->v ->u ->t. The total running time of the algorithm will be O(total number of edges * MAXIMUM FLOW), as in each iteration only one unit of flow will pass through.

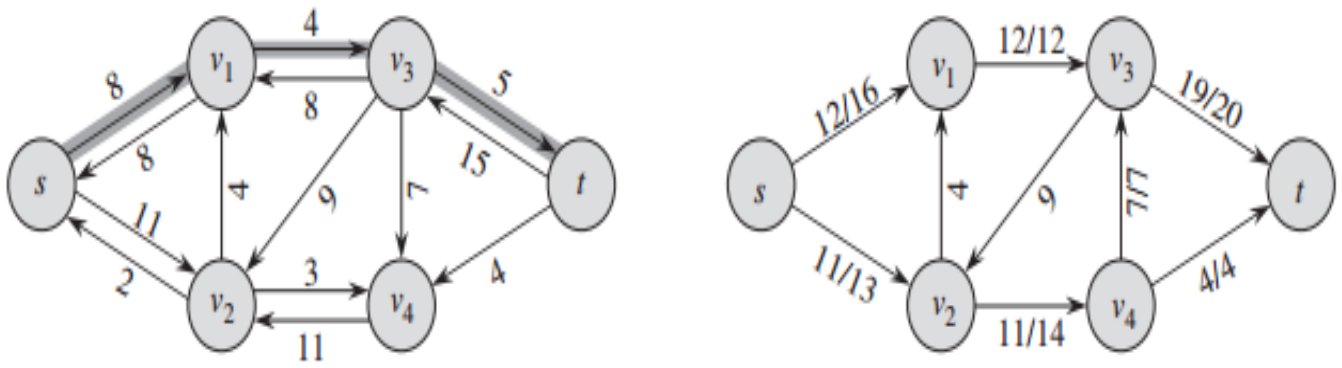


Figure 9:

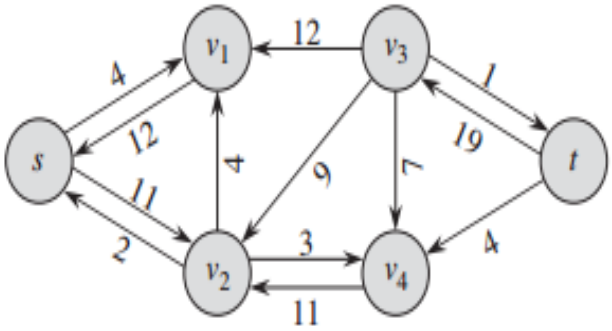


Figure 10:

4 Edmonds-Karp algorithm

Edmonds-karp algorithm is a modified version of fork-fulkerson algorithm. Instead of iterating with DFS Edmonds-karp used BFS. That is, we choose the augmenting path as a shortest path from s to t in the residual network, where each edge has unit distance (weight). We call the Ford-Fulkerson method so implemented the Edmonds-Karp algorithm. We now prove that the Edmonds-Karp algorithm runs in $O(V^2E)$. As in each augmented path the distance between nodes is increasing monotonically.

4.1 Explanation

Figures 12-15 show successive stages of the Edmonds-Karp algorithm, including the 4 augmenting paths selected, while solving a particular maxflow problem. "Real" edges in the graph are shown in black, and dashed if their residual capacity is zero. Green residual edges are the back edges created to allow "undo" of flow on a "real" edge. Each graph containing

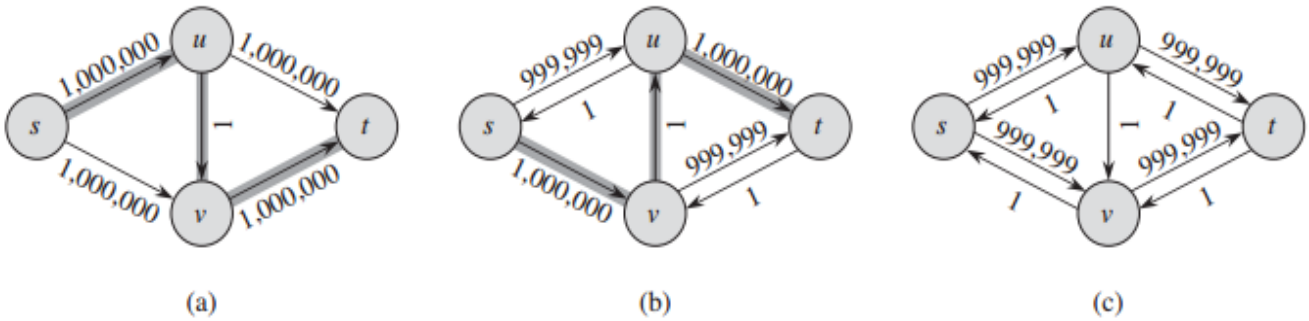


Figure 11:

an augmenting path is drawn twice– first as a "plain" graph, then showing the layering induced by breadth-first search, together with an augmenting path chosen at that stage (light blue). G4 has no remaining augmenting paths (edges from s are saturated); G5 is the resulting max flow, with each edge annotated by "flow" / "capacity". Note how successive augmentations push nodes steadily farther from s, and especially that (undirected) edge a,f is the "critical" edge twice – first in G0, when a is at depth 1 in the BFS tree, and again in G3 when f (not a) is at depth 3, which allows us to undo the "mistake" of sending any flow through this edge. Edge capacities of 10 could be replaced by any value C greater than 1 without fundamentally altering the series of graphs shown. Hence, Ford-Fulkerson (lacking the E-K-D shortest path innovation) might use C augmentations on G0, instead of 4.

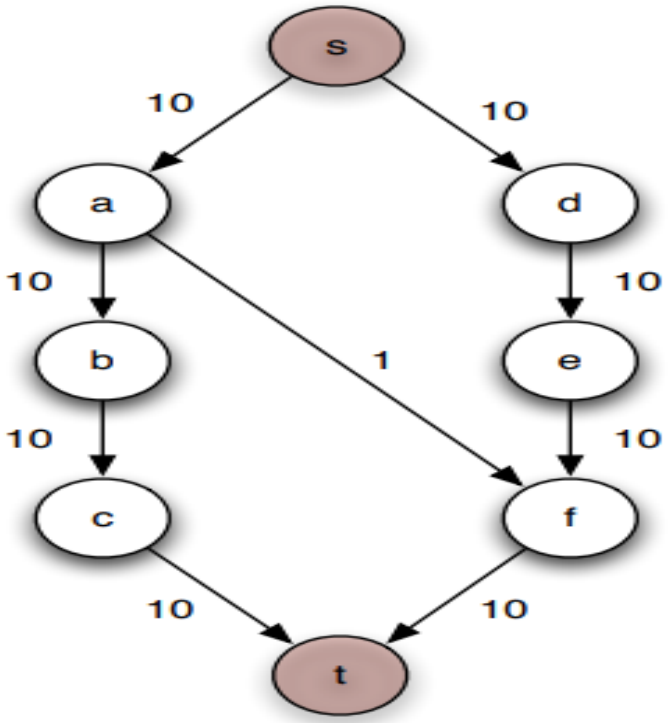


Figure 12:

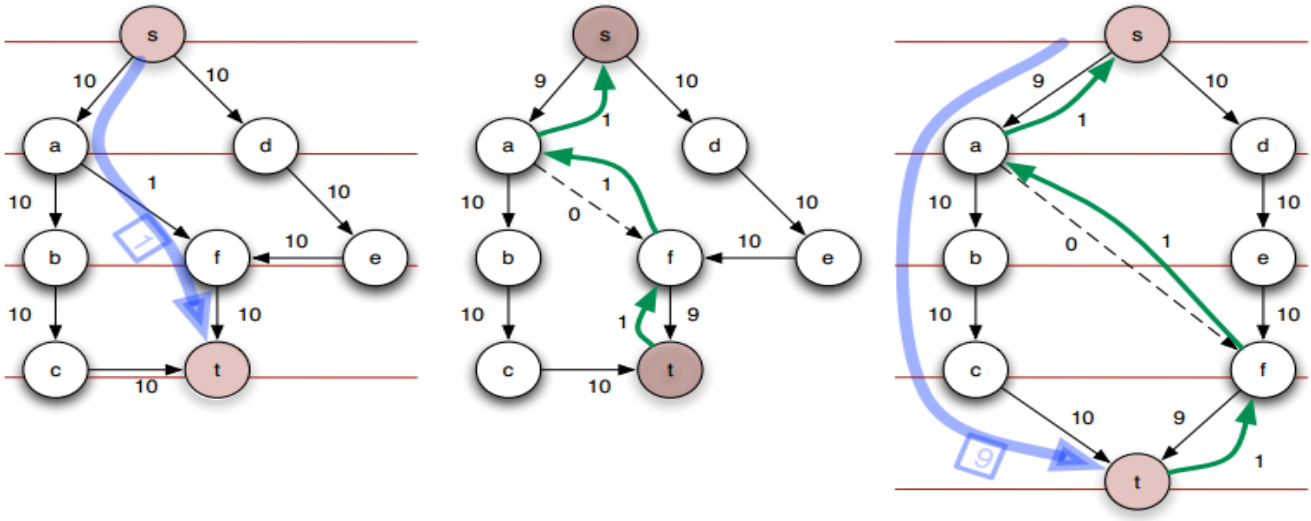

s
a
d
b
f
e
c
t
10
1
9
0
G_0: BFS layering + Aug Path
G_1: 1st Residual Graph
G_1: BFS layering + Aug Path

Figure 13:

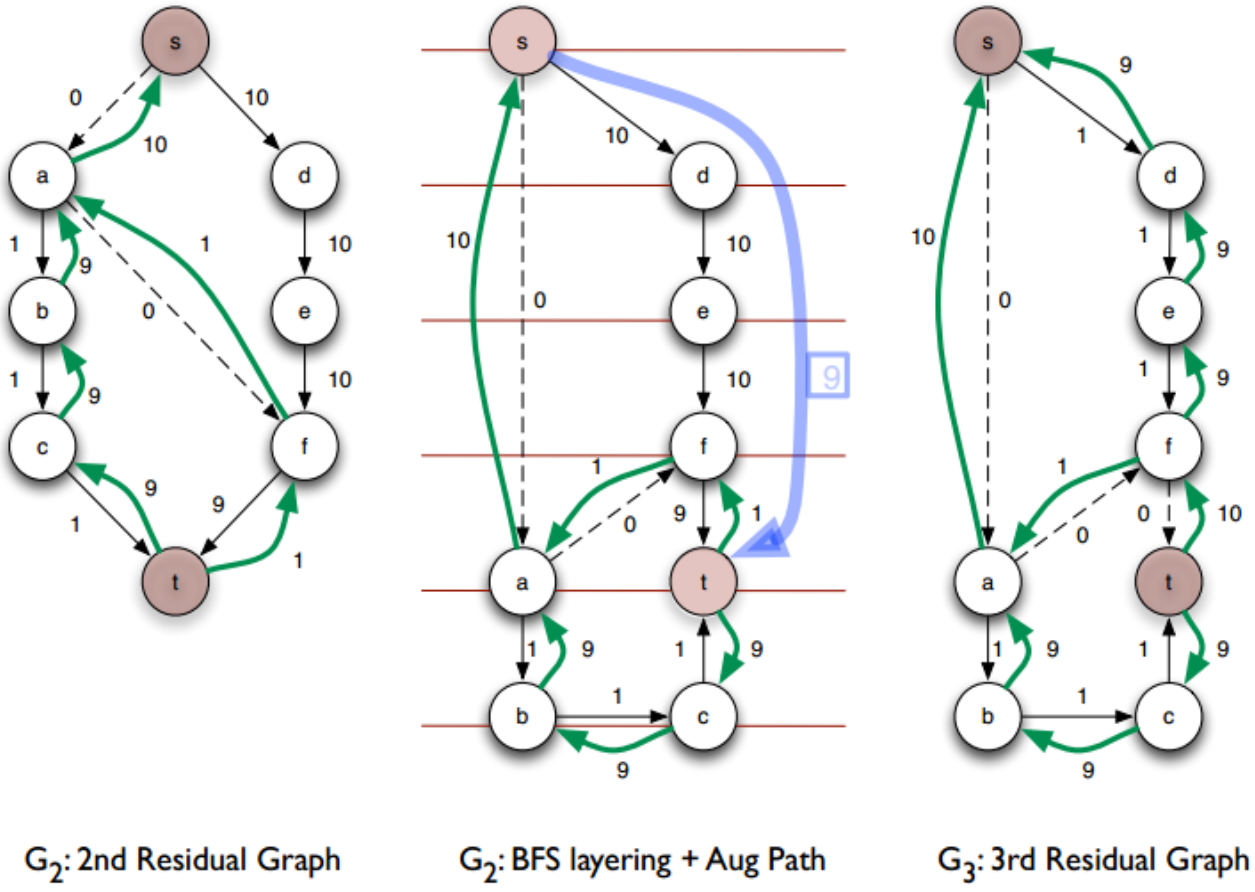

s
a
d
b
e
c
f
t
0
10
1
9
G_2: 2nd Residual Graph
G_2: BFS layering + Aug Path
G_3: 3rd Residual Graph

Figure 14:

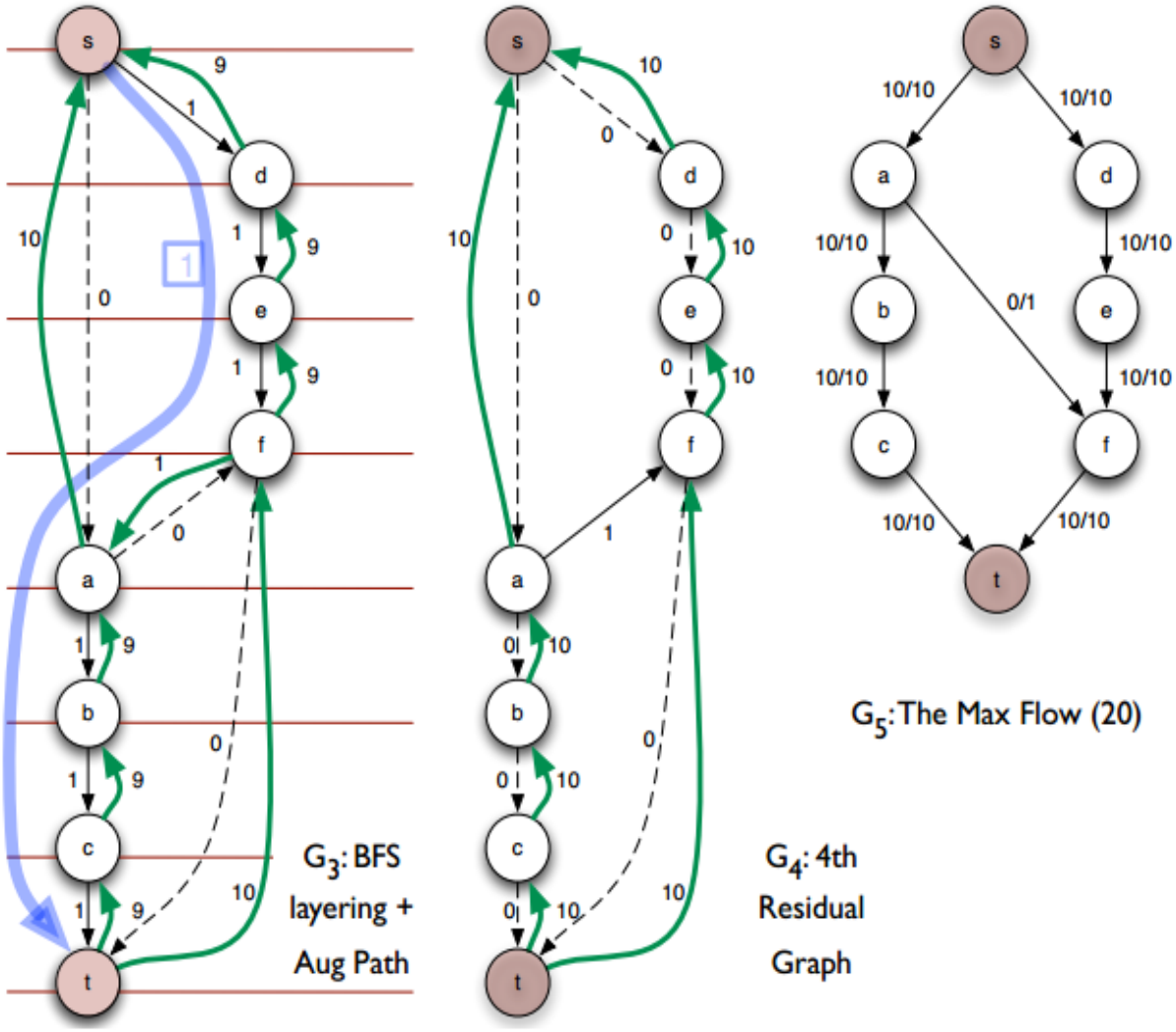

s
d
e
f
a
b
c
t
G3: BFS layering + Aug Path
G4: 4th Residual Graph
G5: The Max Flow (20)
10/10
0/1

Figure 15: