

k

Visualization of Dinic Algorithm

Rahul Ramesh, Vignesh.M, R.Gowrisankar

1 PROBLEM DEFINITION

The Dinic Algorithm aims to solve the commonly known maxflow problem. Maximum flow problem tries to find the largest flow from designated source to sink vertices in a flow network. A flow network is given by a directed graph, with the weights representing the maximum possible capacities along that edge.

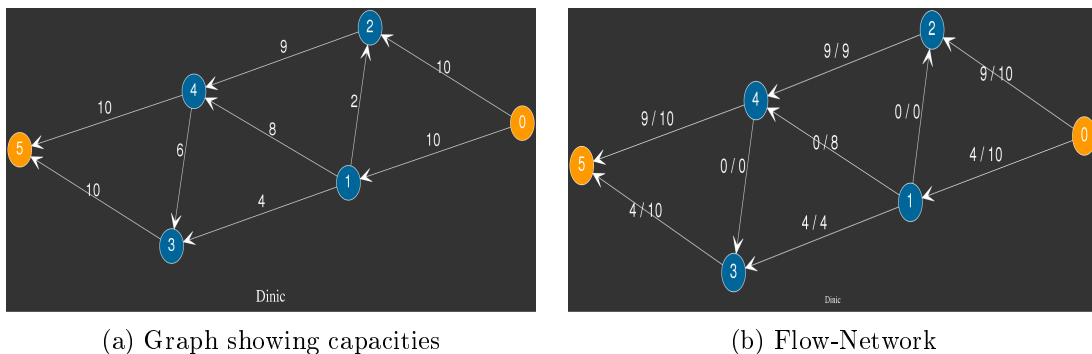


Figure 1.1: Example with Source-Sink flow = 13

A flow-network is a directed graph where the weights along each edge is sandwiched between zero and the maximum capacity. The value of this weight is known as the flow. The Dinic algorithm gives us a flow-network, such that the flow from the source to the sink is maximised. The same problem can be solved using many algorithms like the Ford-Fulkerson Algorithm, Edmonds-Karp Algorithm Push-Relabel Algorithm or the Dinic Algorithm. We will look at the Dinic Algorithm, and in particular the implementation that runs in $\mathcal{O}(V^2E)$ where V and E are the number of vertices and edges in the graph

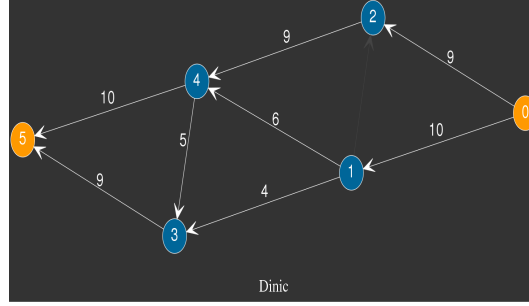


Figure 1.2: Flow-network with Max-flow value after Dinic

Before looking at the algorithm, we will define a few terms like Residual Graph, Level graph and Augmenting Path.

2 DEFINITIONS

2.1 RESIDUAL GRAPH

Consider two vertices V_1 and V_2 . Let the flow from V_1 to V_2 to be F_{12} along the edge E_{12} and flow from V_2 to V_1 to be F_{21} along the edge E_{21} . If no edge exists from one vertex to another, then set the flow value to be 0. Also let C_{12} and C_{21} represent the maximum capacity of edges, E_{12} and E_{21} .

The Residual graph contains all the vertices present in the original graph. For each pair of vertices V_1' and V_2' in the Residual-graph, assign the following:

$$\begin{aligned} F'_{12} &= (C_{12} - F_{12}) + C_{21} \\ F'_{21} &= (C_{21} - F_{21}) + C_{12} \end{aligned} \tag{2.1}$$

Residual graph in essence, builds a graph, representing the maximum possible change in flow from V_1 to V_1 or, V_1 to V_1 constrained by the capacities. This graph is used as a tool to increase or decrease flow values along edges, in order to increase the overall source sink-flow. One can discard all edges with zero flow edges in the Residual graph.

2.2 LEVEL GRAPH

The Level graph is a sub-graph of the residual graph. Split the vertices in the residual graph into various levels. A vertex is said to belong to level 'i' if the shortest distance from the Source vertex is 'i'. After dividing the vertices into levels, retain only those edges, which go from a level 'i' to level 'i+1'.

2.3 AUGMENTING PATH

Given a flow-network and it's corresponding maximum capacities, an augmenting path is a source-sink path, where all the edges have the flow strictly less than the maximum

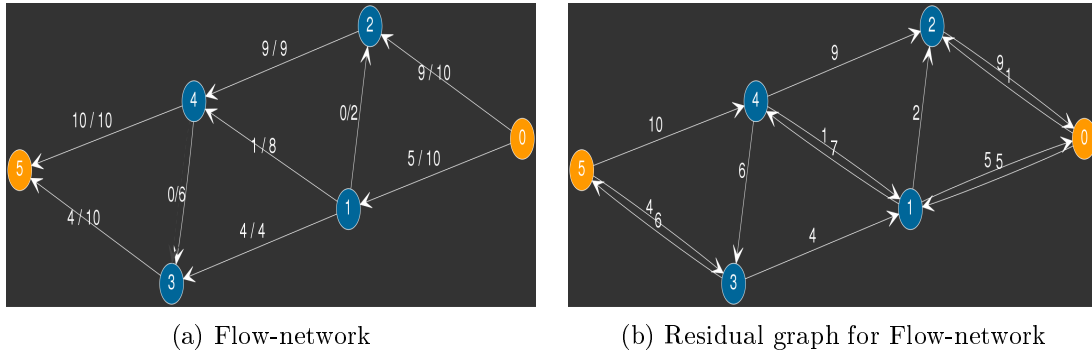


Figure 2.1: Example for Residual graph

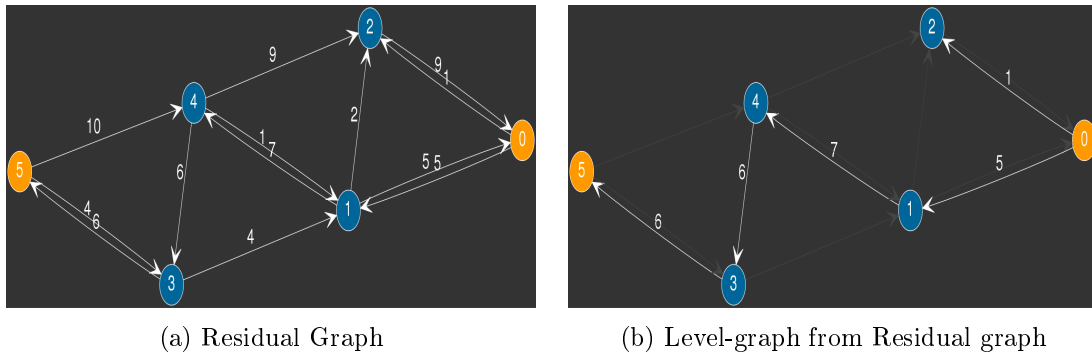


Figure 2.2: Example for Level graph

capacity of the edge. Augmenting path's are used to increase the source-sink flow. If one finds an Augmenting path, then one can increase the flow along that path by $\min(C_{uv} - F_{uv})$ for all edges E_{uv} such that E_{uv} belongs to the Augmenting Path. Note F_{uv} and C_{uv} denote the flow and capacity values. Figure 2.3 highlights an augmenting path, and the flow along the path is increase by 4.

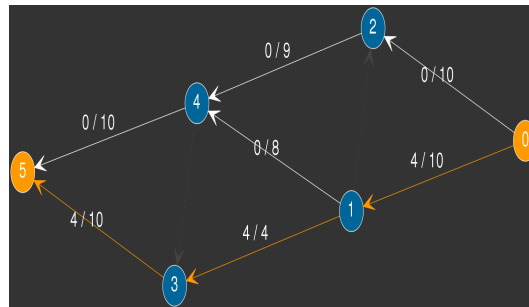


Figure 2.3: Augmenting Path denoted in Orange

2.4 BLOCKING FLOW

Blocking Flow is a flow network where all possible source-sink paths have atleast one saturate edge i.e. $C_{uv} = F_{uv}$ for atleast one edge E_{uv} , along any source-sink path. Alternatively, the graph has no Augmenting Path. A blocking flow need not be a maxflow, but the converse is true.

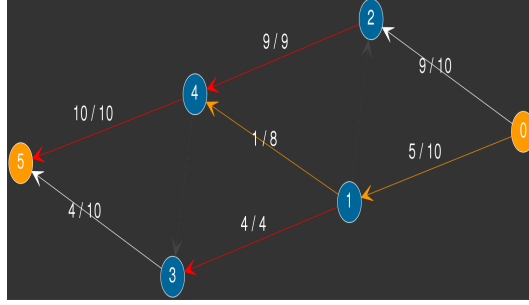


Figure 2.4: Blocking Flow with Red, denoting that the edge is saturated

3 ALGORITHM

3.1 PSUEDO CODE

Algorithm 1 Dinic Algorithm

Data: Graph capacities

Result: Maxflow network and maxflow value

Initialization : Set the flow network values to be zero for all edges

```

for ( $i=1$  ;  $i \leq V$  ;  $i++$ ) do
     $G' = \text{Level}(\text{Residual}(G))$ 
     $G' = \text{BlockingFlow}(G')$ 
    Increment edges in  $G$  with values from  $G'$ 
end
return  $G$ 

```

As mentioned earlier, the Run-time of the algorithm is $\mathcal{O}(V^2E)$. Finding the Level graph or Residual graph takes $\mathcal{O}(E)$ since, it suffices to iterate over all edges in the graph. A blocking flow can be found in $\mathcal{O}(VE)$. Combining these two results, we get the required complexity of the algorithm.

The complexity for the blocking flow is not trivially true. The Blocking Flow routine calls the Modified DFS E times.

$$T(\text{BlockingFlow}) = E \times T(\text{DFSRoutine}) \quad (3.1)$$

Each DFS routine takes $\mathcal{O}(V)$ time to find a source-sink path in each iteration. Additionally, over all the E iterations in the Blocking Flow routine, the DFS routine can delete atmost of E edges.

Note that :

$T(\text{DFS path finding}) = \mathcal{O}(V)$

$T(\text{DFS Deletion over } E \text{ function calls}) = \mathcal{O}(E)$

$$T(\text{BlockingFlow}) = E * T(\text{DFSpathfinding}) + T(\text{DFSDeletion}) \quad (3.2)$$

$$\implies T(\text{BlockingFlow}) = \mathcal{O}(VE) \quad (3.3)$$

Algorithm 2 Blocking Flow

Data: Graph capacities

Result: Blocking Flow of the Graph

Init : Set the flow network values to be zero for all edges and Path_Exists = Yes

```

while Path_Exists=Yes do
    Path = Modified DFS(G)
    /* Find Augmenting Path */
    if No Path found then
        | Path_Exists = No
    else
        | Increase flow along Augmenting Path in G
    end
end
return G

```

Algorithm 3 Modified DFS

Data: Flow Network

Result: Augmenting Path

Do DFS from source

Stop DFS if sink is reached

Delete Vertex is no Source-sink path possible from Vertex

```

if Sink encountered then
    | Augmenting_Path = Path found from DFS
else
    | Augmenting_path = None
    | /* If Sink, never encountered then no Source-Sink path exists */
end
return Augmenting_path

```

3.2 CORRECTNESS

3.2.1 TERMINATION

Firstly, we shall prove that the algorithm always terminates. Each iteration in the Dinic function always, increases the source-sink distance in the level Graph. If this is true, since, the source-sink distance is less than V , termination is ensured

Using proof by contradiction, assume that the source-sink distance does not change, after updating the graph with a blocking flow. Path present in the previous iteration cannot exist in current iteration since,

$$\exists E_{uv}, \text{ such that } F_{uv} = C_{uv},$$

Hence this edge will be removed in the Residual graph and hence the path cannot be present in the next iteration. Hence new edge must be present in the path. Any new edge must be from Level 'i' to Level 'j' where 'i' \geq 'j'. Note that the edges are with respect to the previous iteration. No edges can go to higher levels, by virtue of construction of the level graph. Hence since edges, are constrained by being from Level 'i' to Level 'j' where 'i' \geq 'j', there can exist no source-sink path that is of same length as previous iteration. Hence we see that the algorithm terminates.

3.2.2 CORRECT RESULT