Interior Point Methods and Greedy Algorithms for Network Flow Problems

Eslam Zaher, Mahmoud Yasser

January 2021

Contents

1 Introduction

Network flows is a problem domain that investigates several fields, including computer science, applied mathematics, engineering, management and operations research. The field roots back to the work of Gustav Kirchhoff and other pioneers who systematically analyzed the electric circuits and set up the foundations of network flow theory[1]. Most of this early work was descriptive in nature, and in the context of our project here, we are more interested in optimization questions, that is, if we have alternative ways to utilize the network (i.e send flow), which alternative will be most cost-effective? The apparatus needed for addressing such question is more recent and traced back to the late 1940s and early 1950s when optimization was developed and widely applied by the research and practitioner communities, enabling for scientific and computational inaugurations[1]. In this work, we shall investigate network flow problems with linear programming formulation,and we will use interior point methods and a greedy algorithm (Edmonds-Karp) to solve these problems. In the next sections, we wish to address the following: Problem formulation of Maximum Flow, Shortest Path, and Maximum Bipartite Matching problems, interior point methods and Edmonds-Karp greedy algorithm for efficiently solving such problems, and our discussion on results.

2 Linear Programming Formulation for Network Flows

2.1 Maximum-Flow Problem

The Maximum Flow Problem over a capacitated network, as shown in figure1, is represented by a graph **G(V,E,C)**, where **V** is the set of vertices, **E** is the set of edges, **C** corresponds to the capacities associated to each edge. It aims at maximizing the total flow coming out from a source node to a destination node. In the linear programming formulation of the problem, we have two types of constraints; the conservation of flow constraint or Kirchhoff's law which gurantees that the total flow entering a node equals the total flow leaving that node except for the source and target (sink) nodes, the second constraint guarantees that the capacity on each edge is the maximum flow that can pass in that edge, that is, edge flow can not exceed edge capacity. The linear programming formulation for the Max Flow problem is as follows with the two types of constraints represented by equation 1 and 2 respectively.

$$\text{Maximize } \sum_{s,u} f_{s,u}$$

$$\text{S.t} \begin{cases} \sum_{u:u,v\in E} f_{u,v} - \sum_{u:v,u\in E} f_{v,u} = 0 \quad \forall v \neq s,t \ \ (1) \\ 0 \leq f_{u,v} \leq c_{u,v} \ \ (2) \end{cases}$$

2.2 Shortest Path Problem

The Shortest Path Problem on a graph is the problem of finding a path of minimum cost between two nodes, source and destination. It arises in many routing applications in computer networks and transportation and there exist many heuristic search and greedy methods such as A^* and Bellman–Ford algorithm to solve the problem. In our context here, We are more interested in solving this problem with linear programming methods. We define $f_{u,v}$ to be a binary variable with $f_{u,v} = 1$ if the arc (u to v) is on the shortest path and $f_{u,v} = 0$ otherwise. Each edge (u to v) in the network is associated with a cost or weight $c_{u,v}$. The linear programming formulation for this problem is as

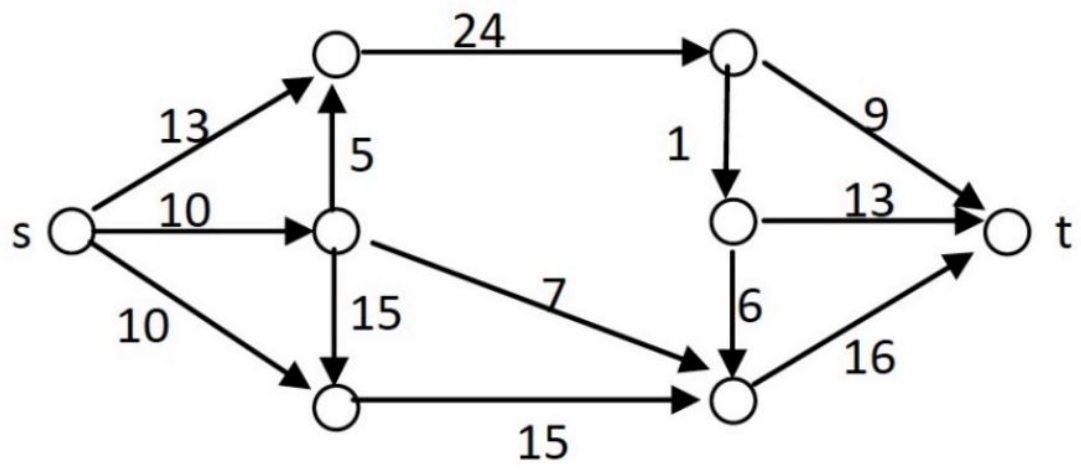


Figure 1: A flow network with edges capacities

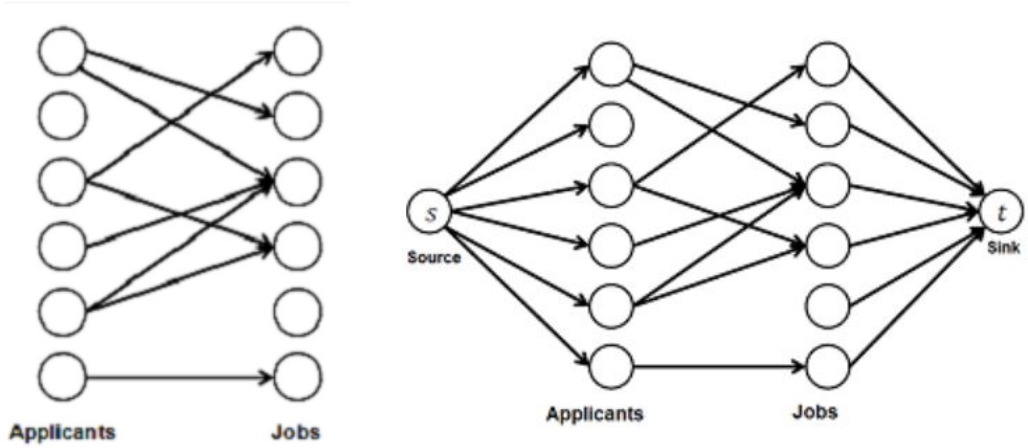
follows:

$$\text{Minimize } \sum_{u,v} c_{u,v} * f_{u,v}$$

$$\text{S.t} \begin{cases} \sum_{v:s,v\in E} f_{s,v} = 1 \quad \text{(starting node)} \\ \sum_{v:v,t\in E} f_{v,t} = 1 \quad \text{(target node)} \\ \sum_{u:u,v\in E} f_{u,v} - \sum_{u:v,u\in E} f_{v,u} = 0 \quad \forall v \neq s,t \quad (1) \\ 0 \leq f_{u,v} \end{cases}$$

2.3 Maximum Bipartite Matching (MBM) Problem

In this problem, We want to maximize the number of matching edges in a bipartite graph **G(V,E)**, shown in figure 2-a. The maximum matching size is maximum size of a set $M \subseteq E$ such that each vertex in the graph has at most one incident edge in M. The problem arises in job matching, advertising, recommender systems, and many other applications. The MBM problem can be formulated as a Max Flow problem using a network flow conversion of the bipartite graph as shown in figure 2-b and can be solved using the same methods used for Max Flow Problems.



(a) Bipartite graph

(b) Network flow

Figure 2: Maximum Bipartite Matching : (a) Bipartite Graph; (b) Network flow conversion

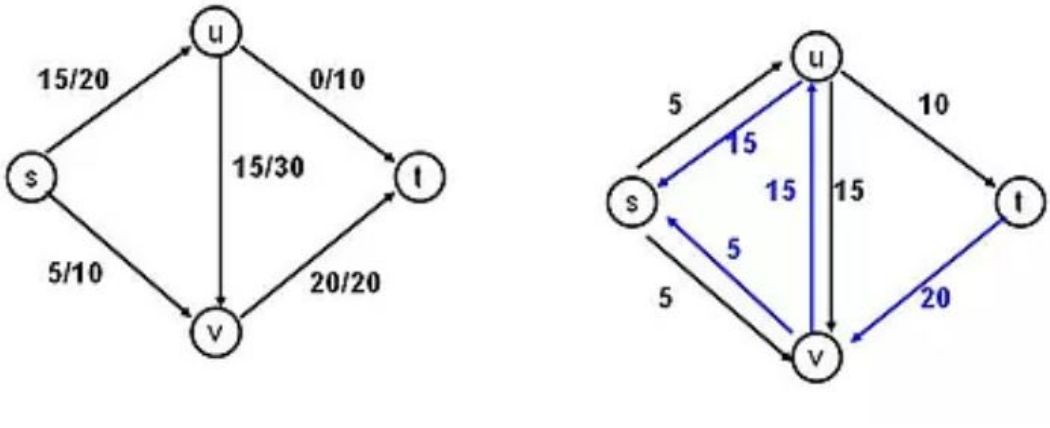
3 Method

3.1 Interior Point Methods

In solving the linear programs mentioned in section 2, we are utilizing our implementations of the fixed and adaptive central path methods, and Mehrotra predictor-corrector algorithm for efficient computations. Our code corpus takes the network flow problem as input and output a solution graph. It extracts the linear program from the input graph, uses one of the algorithms to solve it, and returns the maximum flow value (for max-flow problems) and solution graph.

3.2 Edmonds-Karp Algorithm

Edmonds-Karp algorithm is a polynomial time greedy method with time complexity $\mathbf{O}(|\mathbf{V}||\mathbf{E}|^{\mathbf{2}})$, where $|\mathbf{V}|$ is the number of vertices and $|\mathbf{E}|$ is the number of edges. The algorithm utilizes Breadth First Search (BFS) to find an augmenting path in a residual graph of the network flow. A residual graph, as shown in figure 3-b, has corresponding residual capacities which are calculated as the difference between the capacity and current flow in each forward edge. Figure 3-b shows the residual network for the network in figure 3-a. It is worth noting that if there is an edge (u, v) in the original network with a certain amount of flow on it, then we'll have an opposite edge (v, u) in the residual graph with a capacity equal to that flow. If an edge in the original graph is saturated, that is its flow equals its capacity, then the corresponding edge is omitted from the residual graph as it cannot admit anymore flow. The algorithm uses BFS to find the shortest augmenting path P from the source s to the target t in the residual graph, and each iteration the flow is calculated as the minimum current residual capacity on the path into each backward edge that leads from t to s . The same amount of minimum capacity is subtracted from each forward edge in augmenting path, and each iteration the residual network is updated. We implemented our variant of the Edmonds-Karp algorithm in python, and we are mainly utilizing it for the max-flow problem. Edmonds-Karp is proved by max-flow min-cut theorem to solve the maximum flow problem[2].



(a) Flow network

(b) Residual network

Figure 3: Network flow: (a) Original network flow; (b) Residual network

4 Results and Discussion

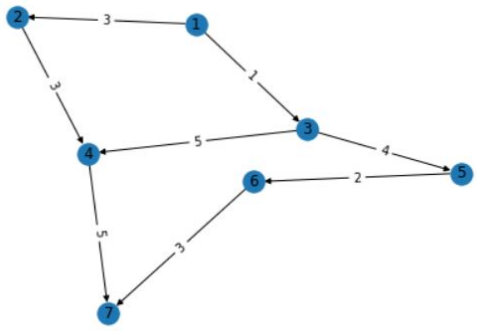
In this section, we are showing results on three instances for both the max-flow and shortest path problems, and one instance for the maximum Bipartite matching problem. We compare our results with the solutions from NetworkX, a python library developed by the Los Alamos National Laboratory for studying graph and networks. In the case of max-flow problems, we also compare our results with the ones resulted from the Edmonds-Karp algorithm mentioned in section 3.2.

Both Edmonds-Karp and interior point methods exhibit polynomial time performance, the difference is that Edmonds-Karp algorithm is a graph-theory based greedy method that is contingent on the properties and structure of the problem under investigation and it is easy to implement, while interior-point methods are more robust, include heavy computations and analysis, and are more difficult to implement, but only depend on the formulation, that is if the problem has a linear (and possibly non-linear) programming formulation, interior point methods are guaranteed to work.

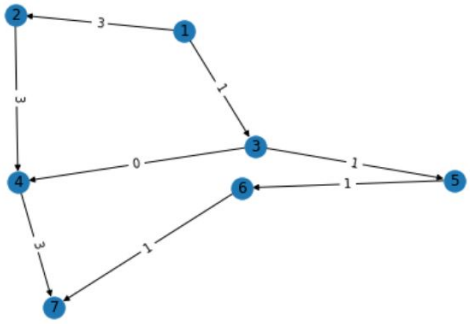
4.1 Max-Flow and Shortest Path Problems

In our implementation of the interior point methods, we added a rounding routine to cast the resulting flows into integer values, this gave us the behaviour of a mixed-integer programming method, with results comparable to that produced by Edmonds-Karp and NetworkX solver, which preserve intrinsically the integer values for flows.

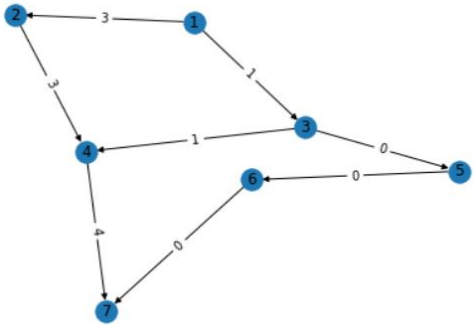
Figure 4 shows a max-flow problem instance solved by different methods. Figure 4-a shows the original capacitated network to be solved for maximum flow. The resulting flow graph solved by interior-point methods(Mehrotra in this case) is shown in figure 4-b and it has a maximum flow of 4. Figure 4-c shows the resulting flow solved by both the Edmonds-Karp algorithm and NetworkX solver. It is worth noting that the solution resulted by Interior point methods is an alternative to the ones resulted by both Edmonds-Karp and NetworkX with the same maximum flow value for all methods. Figure 5 shows another example solved by the same approach. In figure 6, we show our results for the Shortest Path problem. the left figure in 6-a shows the original graph with edge costs, the right figure of 6-b shows the resulting graph in which each edge included in the shortest path in tagged with 1 as a cost. Figure 6-c and 6-d show the shortest path total cost and the evolution of edge flows(costs) over iterations respectively.



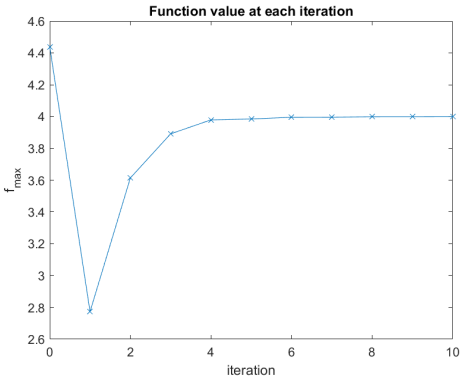
(a) Original Capacitated Graph



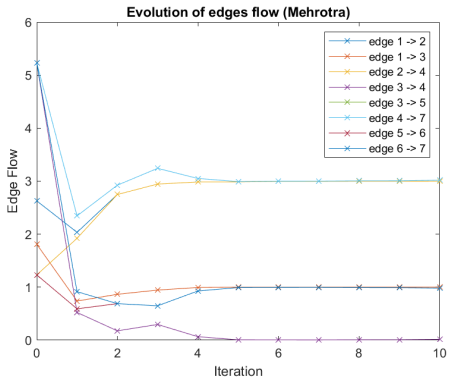
(b) Flow Graph using Mehrotra Method



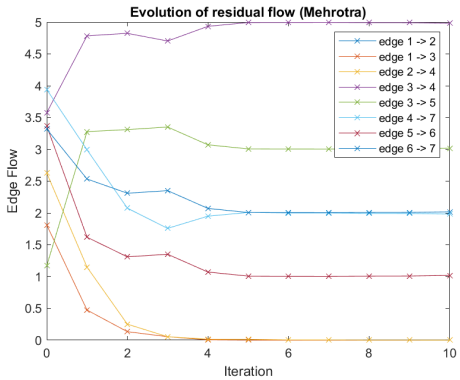
(c) Flow Graph using Edmonds-Karp and NetworkX Solver



(d) Evolution of Flow Value

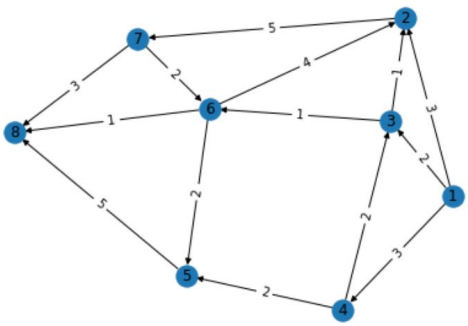


(e) Evolution of Edge Flows

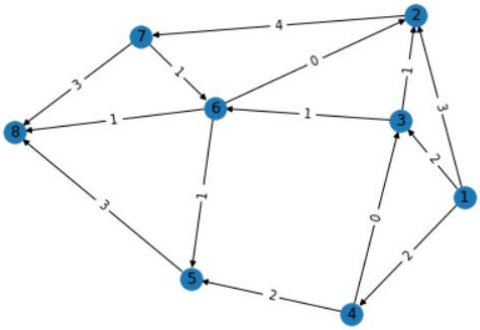


(f) Evolution of Residual Edge Flows : c-f

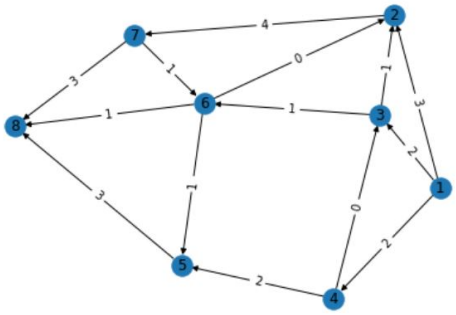
Figure 4: Max Flow Problem : (a) Directed Capacitated Graph; (b) Network Flow Solution using Mehrotra; (c) Flow Graph using Edmonds-Karp and NetworkX solver; (d) Evolution of Flow Value; (e) Evolution of Edge Flows; (f) Evolution of Residual Edge Flows



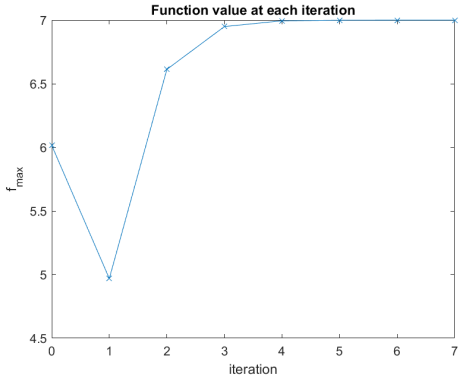
(a) Original Capacitated Graph



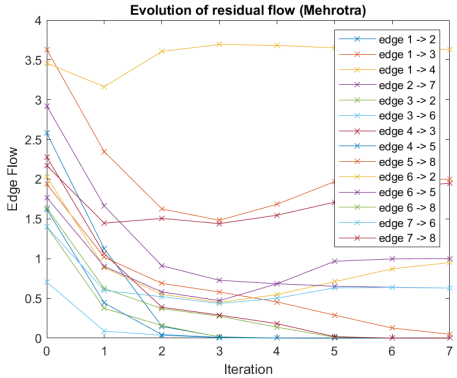
(b) Flow Graph using Mehrotra Method



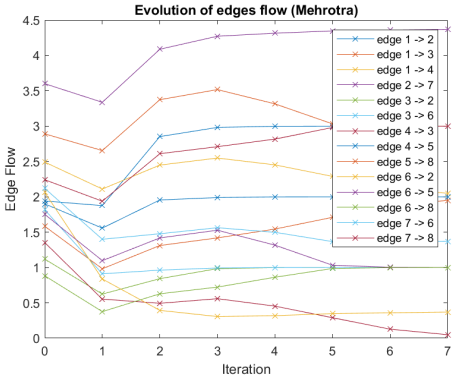
(c) Flow Graph using Edmonds-Karp and NetworkX Solver



(d) Evolution of Flow Value

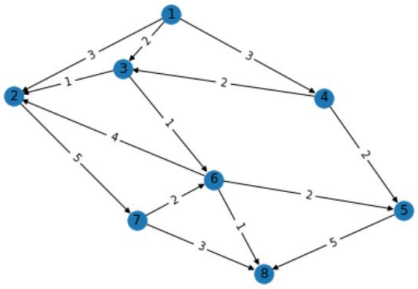


(e) Evolution of Edge Flows

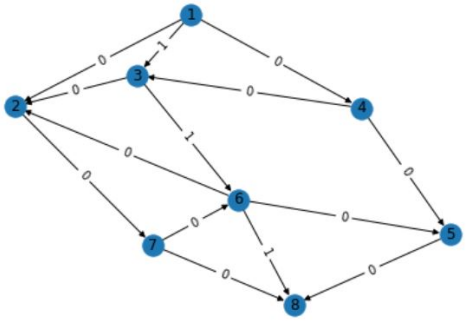


(f) Evolution of Residual Edge Flows

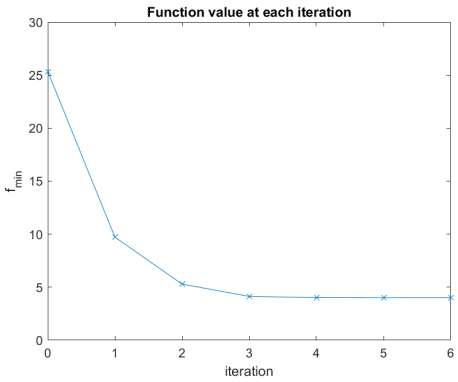
Figure 5: Max Flow Problem : (a) Directed Capacitated Graph; (b) Network Flow Solution using Mehrotra; (c) Flow Graph using Edmonds-Karp and NetworkX solver; (d) Evolution of Flow Value; (e) Evolution of Edge Flows; (f) Evolution of Residual Edge Flows



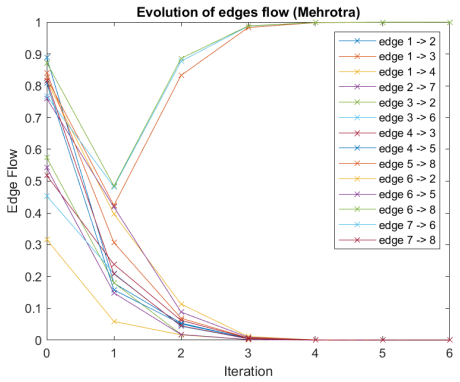
(a) Directed Graph with Edge Costs



(b) Shortest Path Solution



(c) Evolution of Total Path Cost Value

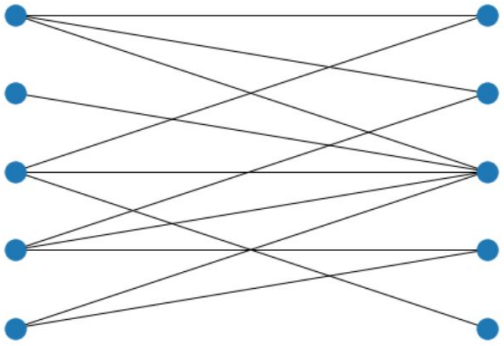


(d) Evolution of Edge Flows(Included Edges

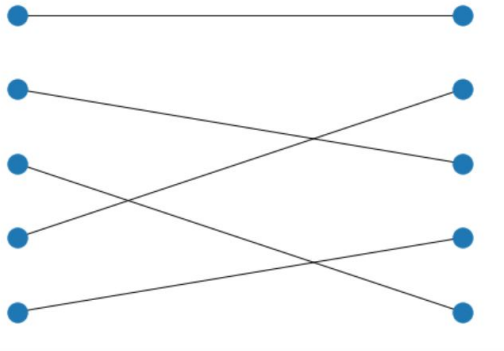
Figure 6: Shortest Path Problem: (a) Directed Graph with Edge Costs; (b)Shortest Path Solution; (c) Evolution of Total Cost Value; (d) Evolution of Edge Flows(Included Edges)

4.2 Maximum Bipartite Matching Problem

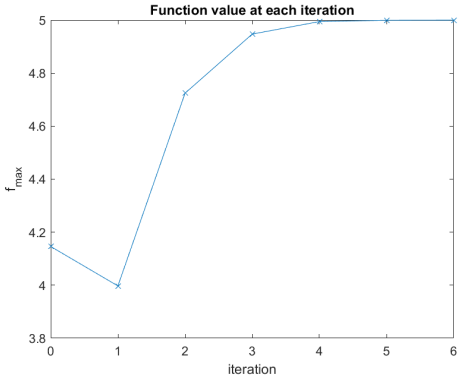
In the maximum bipartite problem, we have two sets of vertices, a right set where each vertex is edged to one or more vertex from a left set. The aim here is to match each vertex from any set to at most one vertex from the other set. It is not guaranteed that each vertex will be matched, thus the maximum bipartite problem is concerned with the maximization of the number of matched edges. The problem is formulated as a max flow problem as mentioned in section 2.3 and is solved in the same way as max flows. Figure 7-b shows the matching solution to the bipartite system in 7-a, while figure 7-c and 7-d show the number of matched edges (seen as maximum flow) and the evolution of edges flow (assigned value : 1 or 0)



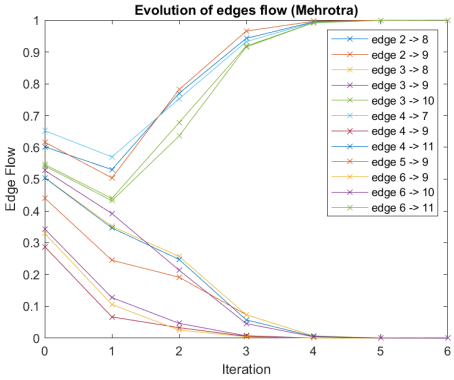
(a) Bipartite Graph



(b) Matching Solution



(c) Evolution of Matching Edges Value



(d) Evolution of Matching Edges

Figure 7: Maximum Bipartite Matching: (a) Bipartite Graph; (b) Matching Solution; (c)Evolution of Matching Edges Value; (d)Evolution of Matching Edges

5 References

[1] James B. Orlin, Ravindra K. Ahuja, and Thomas L. Magnanti, "Network Flows: Theory, Algorithms, and Applications"

[2] Peter Lammich, S. Reza Sefidgar, "Formalizing the Edmonds-Karp Algorithm"