CS4445 – Analysis of Algorithms 2  
Roberto Solis-Oba  
Final Project  
Applications of Max Flow, Min Cut and Maximum Matching Problems  
Daniel Gilbert  
250 697 537

Many real world problems can be described by a diagram consisting of a set of a set of points with lines connecting certain pairs of these points (Bondy & Murty, 1). This type of diagram is known as a graph, the points are known as vertices and the lines that connecting them are known as edges (Golumbic & Hartman, 41). This paper will address graphs on a very basic level to make it more accessible. For those unfamiliar with any concepts addressed within this paper, a thorough outline of basic definitions in Graph Theory is provided by John Essam and Michael Fisher (Essam & Fisher, “Some Basic Definitions”).

The study of graphs and the algorithms that can be utilized on them is an interface between combinatorial mathematics and computer science (Golumbic & Hartman, 41). As a result of the complex theoretical backgrounds that ground work with graphs, it is often difficult to understand their applications to real world problems. This paper is an attempt to breakdown this barrier of understanding. It will discuss some real world applications of some of the major algorithms in graph theory, namely: Max Flow algorithms, Min Cut algorithms, and Maximum Matching algorithms. The first section will discuss Max Flow algorithms, the second Min Cut algorithms and the third Maximum Matching algorithms. It is important to note that the value of the maximum flow is equal to the capacity of the minimum cut (Ray, 163). As such, many of the applications for Max Flow algorithms will be the same as those for Min Cut algorithms. As well, many instances of the Maximum Matching problem can be reduced to the Max Flow problem (Kingsford, 6). And thus many applications of the Max Matching problem will be related to the Max Flow problem. Nonetheless, while inherently related, these algorithmic approaches to graphs still have different conceptualizations and thus lend themselves better to different problems. As such, they shall be addressed separately in this paper.

**Max Flow Applications**

The maximum flow problem is a type of optimization problem that can be applied to directed, weighted graphs, with two special nodes: the source and the sink (Park, 3-4). The graph is directed in the sense that each edge can only transfer data between vertices in one direction. They can be understood as a one way road. The graph is weighted in the sense that each edge has a specific capacity that they can transfer between vertices (Park, 4). Continuing with the road analogy, we can say that each road has a maximum weight that they can handle; a truck can go down the road if it is carrying a weight up to the maximum capacity of that road, but no more than that.

The source is a vertex from which edges are only directed outward, and the sink is a vertex where edges are only directed inwards. The source is the starting point in a directed graph, and the capacity of the edges leaving the source is the absolute capacity that exists throughout the graph. If we see the directed graph as an electrical network, the source is the generator from which all power originates. The capacity of the edges leaving the source represents the maximum amount of power that can be sent from the power source. The sink is the endpoint in a directed graph and the capacity of the edges into the sink represents the total amount of capacity left after travelling through the network. Continuing the electrical network analogy, the sink would be the electric device that utilizes the power sent through the network. The vertices between the source and the sink can be seen as capacitors that temporarily hold the power flowing through the network. For every vertex other than the source and the sink, the incoming flow is equal to the outgoing flow (Park, 4).

The Max Flow problem is to maximize the total amount of flow from the source to the sink subject to the constraints of the edge capacities and the equivalence of incoming and outgoing flow of vertices other than the source and the sink (Park, 4). This problem can model a multitude of theoretical and real world problems. Some of the more real world applications will now be outlined.

*Baseball Elimination*

Every year, millions of baseball fans eagerly watch their favorite team, hoping they will earn a spot in the playoffs and ultimately the World Series (Erickson, 3). Sadly, most teams are “mathematically eliminated” days or even weeks before the regular season ends (Erickson, 3). A team is eliminated if it becomes impossible to win enough games to catch up to the current leader in their division (Erickson, 3). While it is often easy to determine when a team is eliminated, it can sometimes be more subtle and harder to spot (Erickson, 3). A Max Flow algorithm can be utilized to model this problem and determine when a team is eliminated.

Our input consists of two arrays W[1 .. n] and G[1 .. n, 1 .. n], where W[i] is the number of games team i has already won, and G[i, j] is the number of upcoming games between teams i and j (Erickson, 4). We want to determine whether team n (our favorite team) can end the season with the most wins (possibly tied with other teams) (Erickson, 4). We want to assign a winner to each game, so that team n comes in first place (Erickson, 4). We Let R[i] = ∑ j G[i, j] denote the number of remaining games for team i (Erickson, 4). We will assume that team n wins all R[n] of its remaining games (Erickosn, 4). Then team n can come in first place if and only if every other team i wins at most W[n] + R[n] − W[i] of its R[i] remaining games (Erickson, 4). Thus, our team can only win if there is a possible result where no team in the division has more wins than our team has if it wins all of its upcoming games.

We can model this problem by building a bipartite graph, whose nodes represent the games and the teams (Erickson, 4). We have n/2 game nodes g i,j , one for each pair of teams (excluding our favorite) that can play each other, and n − 1 team nodes ti , one for each team in the division other than our favorite team (Erickson, 4). For each pair of teams (i, j), we add edges gi,j -> ti and gi,j -> tj with infinite capacity (Erickson, 4). These edges represent the transfer of wins to remaining teams as a result of the games between these teams. We add a source vertex s and edges s -> gi,j with capacity G[i, j] for each pair i, j (Erickson, 4). These edges represent the number of remaining games between teams i and j (Erickson, 4). Finally, we add a sink node t and edges ti -> t with capacity W[n] − W[i] + R[n] for each team i (Erickson, 4). These edges represent the number of wins attained by a team given that we limit their potential wins to ensure that no team ends up with more wins than our favorite team.

Team n can end the season in first place if and only if there is a feasible flow in this graph that saturates every edge leaving s (Erickson, 5). This means that our team can come out on top, only if the results of the remaining matches between all the other teams can result in them having less wins than our team, assuming that our team won all of its games. If a flow from the source to the sink exists that fully saturates the edges from the source, this means that every game is accounted for within the limitations we’ve proposed (i.e. that no team in the division ends with more win than our team). If the edges from the source can’t be fully saturated, it means that there’s no way our proposed limitations can account for every remaining game. For example, our team may be down by 27 wins, but if every other team loses the next 27 games and our team wins the next 27 games, it will come out on top (Erickson, 4). However, this may not be possible because some of the remaining teams will have to play each other, so there’s no way every team can lose every one of their remaining games (Erickson, 4).

So, to decide whether our favorite team can win, we construct the flow network, compute a maximum flow, and report whether that maximum flow saturates the edges leaving s (Erickson, 5). The flow network has O(n2 ) vertices and O(n2) edges, and it can be constructed in O(n2 ) time (Erickson, 5). Using Orlin’s algorithm, we can compute the maximum flow in O(V E) = O(n4) time.

Below is a representation of a potential case for this problem, derived from the 1996 American League East standing (Erickson, 5). The table shows the current win-loss ratio between teams in a division as well as the remaining games between teams in this division. Our desired team will be the Detroit tigers. The graph will represent the flow network as described above corresponding to the 1996 American League East standings.

. The total capacity of the edges leaving s is 27 (there are 27 remaining games), but the total capacity of the edges entering t is only 26 (Erickson, 5). So the maximum flow has value at most 26, which means that there is no way for the other teams to lose all 27 of their remaining games, and thus Detroit is mathematically eliminated (Erickson, 5).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Team | Won-Lost | Games Left | NYY | BAL | BOS | TOR | DET |
| New York Yankees | 75-59 | 28 |  | 3 | 8 | 7 | 3 |
| Baltimore Orioles | 71-63 | 28 | 3 |  | 2 | 7 | 4 |
| Boston Red Sox | 69-66 | 27 | 8 | 2 |  | 0 | 0 |
| Toronto Blue Jays | 63-72 | 27 | 7 | 7 | 0 |  | 0 |
| Detroit Tigers | 49-86 | 27 | 3 | 4 | 0 | 0 |  |

Source

NYY  
BAL

NYY  
BOS

NYY  
TOR

BAL  
BOS

BAL  
TOR

NYY

BAL

BOS

TOR

Sink

3

8

7

2

7

1

5

7

13

Thus, it has clearly been shown how the Baseball Elimination problem can be modeled by a max flow algorithm. We assume our desired team wins all of its upcoming games, and then create a flow network for a potential result of the games between all the other teams in the same league. The edges to the sink have their capacities limited by the total number of games the corresponding team can win without surpassing the number of win our desired team will have if they win all upcoming games. If there is a way to fully saturate the edges from the source within these limitations, then there is a way our team can win. Otherwise, there is no way our team can come out on top given the current standings. Given the example provided, it turns out the maximum flow is 26, when the maximum saturation of edges from the source is 27. As such, there is no way for the outcome of the remaining games in the league to result in a victory for our desired team.

This example clearly illustrates a real world application for Max Flow algorithms that will be relatable to the average person. Although some understanding of computer science and mathematics might be required to model this problem, this example should be understandable to a less technically inclined audience. This example serves to show the real world applicability of Graph Theory and Network Flow algorithms.

*Matrix Rounding*

Consider an i x j matrix D = {dij} of real numbers with row i sums Ri and column j sums Cj (Wayne, “Max Flow Applications” 64). We want to round the values of each spot in the matrix dij, as well as the row sums Ri and column sums Cj up or down to integer values, so that the sum of rounded elements in each row/column equal the rounded values in the row sums and column sums (Wayne, “Max Flow Applications” 64). This mathematical endeavor has direct applications to the publishing of census data (Wayne, “Max Flow Applications” 64). Two tables representing original data and a possible rounding are depicted below.

|  |  |  |  |
| --- | --- | --- | --- |
| 3.14 | 6.8 | 7.3 | 17.24 |
| 9.6 | 2.4 | 0.7 | 12.7 |
| 3.6 | 1.2 | 6.5 | 11.3 |
| 16.34 | 10.4 | 14.5 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | 7 | 7 | 17 |
| 10 | 2 | 1 | 13 |
| 3 | 1 | 7 | 11 |
| 16 | 10 | 15 |  |

Original Data Possible Rounding

To represent this as a Max Flow problem we must represent the data in graph form. We create a vertex i, to correspond to each row i, and create a node j’ to correspond to each column j (Suri, “Network Flows” 5). We also create two additional nodes for the source and sink (Suri, “Network Flows” 5). We then create an edge connecting the row vertices to the column vertices (i, j’) for each matrix entry dij, an edge connecting the source to each row vertex (s, i) for each row-sum Ri, and an edge connecting the sink to each column vertex (j’, t) for each column-sum Cj (Suri, “Network Flows” 5). The lower and upper bounds for each edge correspond to the values obtained by rounding down and rounding up the corresponding table value (Suri, “Network Flows” 5). A consistent matrix rounding exists if and only if there is a feasible flow in the network corresponding to the edge bounds (Suri, “Network Flows” 5). A network for the original data and a flow in the network representing the possible rounding are depicted below.

Source

1

2

3

1’

2’

3’

Sink

(17,18)

(12,13)

(11,12)

(3,4)

(6,7)

(7,8)

(9,10)

(2,3)

(0,1)

(3,4)

(1,2)

(6,7)

(16,17)

(10,11)

(14,15)

Source

1

2

3

1’

2’

3’

Sink

(17)

(13)

(11)

(3)

(7)

(7)

(10)

(2)

(1)

(3)

(1)

(7)

(16)

(10)

(15)

The feasible flow corresponding to a possible rounding shown in the second table is depicted in the second graph. The flow is a max flow, as it represents the maximum flow that can be pushed from the source to the sink. It has thus been clearly demonstrated that the Matrix Rounding problem can be represented and solved through application of Max Flow algorithms. This application is employable by anyone, regardless of their background in mathematics and thus should clearly demonstrate the real world applicability of Max Flow algorithms.

**Min Cut Applications**

The minimum cut problem is also an optimization problem that can be applied to directed, weighted graphs (Wayne, “Max Flow, Min Cut” 4). It is inherently connected to the Max Flow problem as was noted above; the value of the maximum flow is equal to the capacity of the minimum cut. Thus many of the problems discussed in this section can be reformatted as a Max Flow problem. However, this section will focus on problems that are more intuitively understood as Min Cut problems.

A cut is a partitioning of all the nodes in a graph into two groups S and T, such that the source is in the S group and the sink is in the T group (Wayne, “Max Flow, Min Cut” 7). The capacity of a cut is the sum of the weights of the edges connecting the two groups S and T (Wayne, “Max Flow, Min Cut” 7). The Min Cut problem is to find a cut of a graph that has the minimum capacity possible (Wayne, “Max Flow, Min Cut” 7). This paper will now analyze some of the real world problems this algorithm can be utilized to address.

*Image Segmentation*

The image segmentation problem is a central problem in image processing (Wayne, “Max Flow Applications” 71). The problem is to identify and separate objects in the foreground from the background (Wayne, “Max Flow Applications” 71). We can represent this as a Min Cut problem by first representing the image as a graph.

The vertices in the graph will correspond to the pixels in the picture and the edges of the graph will connect pairs of neighboring pixels (Wayne, “Max Flow Applications” 72). We assign properties F(v) and B(v) to each vertex, where F(v) is the likelihood that pixel v is in the foreground, and B(v) is the likelihood that pixel v is in the background (Wayne, “Max Flow Applications” 72). If F(v) is greater than B(v) than we prefer to label the specified pixel as a part of the foreground (Wayne, “Max Flow Applications” 72). We also assign a property P(v,w) to the edges connecting two vertices that serves as a separation penalty for labeling one edge as foreground and the other as background (Wayne, “Max Flow Applications” 72). If many neighbors of a given edge v are in the foreground, then we prefer to label v as foreground (Wayne, “Max Flow Applications” 72). We define a source which corresponds to the foreground and a sink which corresponds to the background (Wayne, “Max Flow Applications” 74).

To solve the problem we want to separate the vertices into a foreground group which includes the source, and a background group that includes the sink (Wayne, “Max Flow Applications” 74). The goal is to find a partition (S, T) that minimizes: the sum of likelihoods that a pixel is in the foreground among pixels labeled as part of the background, the sum of likelihoods that a pixel is in the background among pixels labeled as part of the foreground, and the sum of separation penalties among those edges that are part of the partition (Wayne, “Max Flow Applications” 73).

There are a variety of ways of determining the likelihood that a given pixel belongs to the background or foreground, but these methods extend beyond what can be appropriately addressed in this paper. Nonetheless, it is hopefully clear that the image segmentation problem can be addressed using Min Cut algorithms, and that this represents a real world application of this methodology.

*Improving Power Networks*

Growing demand for electricity along with deregulation of electricity markets has led to heavy stress on power networks (Duong et al., 929). As a result, existing infrastructure needs to be improved to increase loadability (Duong et al., 929). Installation of Flexible Alternating Current Transmission Systems (FACTS) can be a good way of improving system loadability without building more power lines (Duong et al., 929). FACTS devices can increase transmission capacity and power flow control flexibility and rapidity (Duong et al., 929). However, the effectiveness of FACTS systems is heavily dependent on the location in which they are placed (Duong et al., 929). In order to determine the best place for FACTS devices, engineers must locate the ‘bottleneck’ of the power system (Duong et al., 929). The bottleneck is the first place where congestion occurs in the power system (Duong et al., 929). By reducing the effect of the bottleneck, the overall effectiveness of the power system can be greatly increased (Duong et al., 929). Determining the location of a bottleneck in a power system can be achieved using a Min Cut algorithm (Duong et al., 929).

In order to locate the bottleneck, a power system must first be modeled as a graph (Duong et al., 930). A power system can be modeled as a set N of nodes and a set A of arcs with capacity uij that shows the maximum amount that can flow between node i and j (Duong et al., 930). A virtual source and sink are added to the graph representing the generators and loads respectively (Duong et al., 930). Each line out of the virtual source has a maximum flow that represents the generation of the connected node, and each line into the virtual sink represents the load demanded by the connected node (Duong et al., 930).

Once the graph is formed, the min-cut is found and the bottleneck along this min-cut is determined (Duong et al., 930). The min-cut separates the graph into source and sink portions along the lines of minimum capacity (Duong et al., 930). By identifying the impedance branch along the min cut, (the branch that functions as a limiting factor), the bottleneck of the power network can be determined (Duong et al., 930). By installing the FACTS device along the identified bottleneck the overall effectiveness of the power system can be increased to the maximum extent possible, and thus power companies can get the best bang for their buck (Duong et al., 930).

Thus it is clear to see how Min Cut algorithms can be utilized to determine the location for FACTS devices and improve the efficiency of power systems. By locating the limiting factor within the power system, it is clear where resources are best allocated to improve this system. This clearly represents another important real world use of Min Cut algorithms.

**Maximum Matching**

A matching M of a graph G is a subset of the edges in G, such that no vertex is incident to more than one edge in M (Kavathekar, 1). Intuitively, we can say that no two edges in M have a common vertex (Kavathekar, 1). A maximum matching is the matching of G with the largest possible number of edges (Kavathekar, 1). If every edge in the graph is matched, this is a perfect matching (Kavathekar, 1).

Maximum matchings are most often applied to bipartite graphs: graphs in which the set of vertices can be divided into two disjoint subsets X and Y, such that every edge of the graph has one end point in X and one in Y (Shuchi, 1). As was noted earlier, the task of finding a maximum matching can be reduced to the maximum flow problem (Shuchi, 2). This section of the paper will outline some real world applications of the maximum matching problem.

*Job Scheduling*

The job scheduling problem is concerned with scheduling a set of jobs on a machine (Shuchi, 3). Want to find the largest number of jobs that can possible be scheduled (Shuchi, 3). Let J = {j1, j2, … , jn} represent the set of jobs, and let T = {t1, t2, … , tn} represent slots available on the machine where these jobs can be performed (Shuchi, 3). Each job J has a set of valid slots Sj corresponding to the set T where it can be scheduled (Shuchi, 3). No two jobs can be scheduled at the same time (Shuchi, 3).

This problem can be represented as a bipartite maximum matching problem (Shuchi, 3). For every job create a node in X, and for every timeslot create a node in Y (Shuchi, 3). For each timeslot T in Sj, create an edge from J to T (Shuchi, 3). The maximum matching of this bipartite graph is the largest set of jobs that can be scheduled (Shuchi, 3).

*Roommate Assignment*

The roommate problem attempts to pair potential students with each other that will make the best possible roommates (Suri, “Matching” 2). Students are arbitrarily divided into two groups X and Y. Students fill out their desired requirements for potential roommates such as: non-smoker, quiet, partier, friendly, no-pets, etc. Edges are created between students in the two groups who meet each other’s requirements (Suri, “Matching” 2). The maximum matching of this bipartite graph is the most efficient matching of potential roommates that can be achieved. Remaining students will either have to be paired with less ideal roommates or assigned to single rooms

Matching problems of this kind tend to be too similar to warrant further or more detailed examples. Once matching problems become more complicated, they tend to be best addressed through Max Flow implementations. Below is an example of such a problem.

*Vacation Days*

Consider a hospital that has n doctors, each with a set of vacation days that work for them (Shuchi, 3). There are K vacation periods, each spanning multiple contiguous days (Shuchi, 3). Let Dj be the set of days included in the jth vacation periods (Shuchi, 3). Need to maximize the assignment of doctors to days (one doctor per day) under the following constraints: Each doctor has a capacity Ci which is the maximum total number of days they can be scheduled; For every vacation period, any given doctor is scheduled at most once (Shuchi, 3).

We will solve this problem by network flow (Shuchi, 3). As was done earlier, for every doctor i we create a node ui and for every vacation day j we create a node vj (Shuchi, 3). We add a directed edge from start node s to ui and from vj to sink t (Shuchi, 3). Other edges and capacities are added as follows to satisfy the above constraints:   
• The doctor capacities are modeled as capacities of the edge from the source to the vertex corresponding to the doctor (Shuchi, 3).  
• The capacities of edges from nodes vj to the sink t are all set to 1, to represent choosing one doctor for each vacation day (Shuchi, 3).  
• To prevent the doctor being scheduled more than once in a vacation period, we introduce intermediate nodes. For any doctor i and a vacation period j, we create an intermediate node wij . We create an edge with unit capacity from ui to wij . For every day in the vacation period that the doctor is available, we create an edge from wij to the node corresponding to that day with unit capacity (Shuchi, 3).

Let us see if an integral flow through the graph produces a valid schedule (Shuchi, 4). Since the edge connecting s to the node corresponding to the doctor has the capacity equal to the total availability of the doctor, the flow through the doctor node cannot exceed it (Shuchi, 4). So the first constraint is satisfied. From any doctor to any vacation period the flow is at most one, since the intermediate node has only one incoming edge of unit capacity (Shuchi, 4). This makes sure that the second criterion is met. So the flow produced satisfies all the constraints. If k is the size of an integral flow through the graph, then the total number of vacation days which have been assigned is also k, since the edges which connect the nodes corresponding to the vacation days to the sink node have unit capacity each (Shuchi, 4). So the size of the scheduling is equal to the size of the flow (Shuchi, 4). From this we can conclude that the largest valid scheduling is produced by the maximum flow (Shuchi, 4).

This clearly demonstrates how more complicated cases of matching can be represented as Max Flow problems. Since maximum matching problems can be reduced to Max Flow problems, it is generally in the best interest of algorithm designers to make use of this reduction for more complicated problems. Nonetheless, it is still useful to be aware of the Maximum Matching representation in order to take advantage of its simplicity when more simple problems present themselves.

**Conclusion**

This paper has clearly identified some real world applications for Graph Theory and Network Flow algorithms more specifically. Max Flow, Min Cut, and Maximum Matching algorithms have a variety of applications which extend far beyond the realms of combinatorial mathematics and computer science. It was shown that Max Flow algorithms can be used to address Baseball Elimination and Matrix Rounding problems. It was shown that Min Cut algorithms can be used to address image segmentation and improving of power network. Finally, it was shown that Maximum Matching algorithms can address simple matching problems like job scheduling and roommate assignment. However, more complex matching problems appear to still be best implemented as Max Flow problems.

This paper has addressed the most simple implementations of these algorithms, however more complex implementations can be utilized for things like dynamic summarization of video (Angadi & Naik, “Dynamic Summarization”) and sign language recognition (Tolba et al., “Arabic Sign Language”). These complex implementations are beyond the scope of this paper, but certainly warrant further research for those interested in real world applications of Graph Theory.

The interconnected nature of the Graph algorithms addressed in this paper should also have become clear. Ford-Fulkerson clearly demonstrated the inherent connection between the Min Cut and Max Flow problem. As well, the final section of this paper clearly demonstrated that Maximum Matching problems can be represented as Max Flow problems.

Overall, it should be clear that Graph Theory has an almost endless number of real world applications. The interdisciplinary nature of graph algorithms shows the importance of applying computer science methodology to other disciplines. As computers continue to become increasingly integral to our modern world, it seems inevitable that scientists in every discipline will also have to be computer scientists. This paper hopefully serves as a stepping stone in this progression.

Works Cited

Angadi, Shanmukhappa, and Vilas Naik. "Dynamic Summarization of Video Using Minimum Edge Weight Matching in Bipartite Graphs." *International Journal of Image, Graphics and Signal Processing IJIGSP* 8.3 (2016): 9-18. Print.

Bondy, J. A., and U. S. R. Murty. *Graph Theory with Applications*. New York: American Elsevier Pub., 1976. Print.

Duong, Thanhlong, Yao Jiangang, and Vietanh Truong. "Application of Min Cut Algorithm for Optimal Location of FACTS Devices considering System Loadability and Cost of Installation." *International Journal of Electrical Power &amp; Energy Systems* 63 (2014): 979-87. Print.

Erickson, Jeff. "Applications of Maximum Flow." Lecture 24: Applications of Maximum Flows. University of Illinois, Urbana–Champaign. *Algorithms, Etc.* Web. 11 Apr. 2016.

Essam, John W., and Michael E. Fisher. "Some Basic Definitions in Graph Theory." *Reviews of Modern Physics Rev. Mod. Phys.* 42.2 (1970): 271-88. Print.

Golumbic, Martin Charles., and Irith Ben-Arroyo. Hartman. *Graph Theory, Combinatorics, and Algorithms: Interdisciplinary Applications*. New York: Springer, 2005. Print.

Kavathekar, Paritosh. "Maximum Matching." Cs105 - Maximum Matching. Dartmouth College, Hanover. 2005. *CS105 - Algorithms*. Web. 11 Apr. 2016.

Kingsford, Carl. "Maximum Bipartite Matching." CMSC 451: Maximum Bipartite Matching. University of Maryland, College Park. *University of Maryland Department of Computer Science*. Web. 11 Apr. 2016.

Park, Jaehyun. "Network Flow Problems." CS 97S1 - Network Flow Problems. Stanford University, Stanford. *CS 97S1*. Web. 11 Apr. 2016.

Ray, Santanu Saha. *Graph Theory with Algorithms and Its Applications: In Applied Science and Technology*. New Delhi: Springer, 2013. Print.

Shuchi, Chawla. "Applications of Network Flow." Lecture 5: Applications of Network Flow. University of Wisconsin, Madison. *CS787 - Advanced Algorithms*. Web. 11 Apr. 2016.

Suri, Subhash. "Network Flows." CS 231 - Network Flows. University of California, Santa Barbara, Santa Barbara. *UCSB - CS231*. Web. 11 Apr. 2016.

Suri, Subhash. "Assignment and Matching." CS 231 - Assignment and Matching. University of California, Santa Barbara, Santa Barbara. *CS 231: Topics in Combinatorial Algorithms*. Web. 11 Apr. 2016.

Tolba, M. F., Ahmed Samir, and Magdy Aboul-Ela. "Arabic Sign Language Continuous Sentences Recognition Using PCNN and Graph Matching." *Neural Comput &amp; Applic Neural Computing and Applications* 23.3-4 (2012): 999-1010. Print.

Wayne, Kevin. "Maximum Flow Applications." COS 423 - Theory of Algorithms. Princeton University, Princeton. 2001. *COS 423 - Theory of Algorithms*. Web. 11 Apr. 2016.

Wayne, Kevin. "Max Flow, Min Cut." COS 226 - Max Flow, Min Cut. Princeton University, Princeton. 2004. *Algorithms and Data Structures*. Web. 11 Apr. 2016.