GRAPH THEORY: NETWORK FLOW

Vrushali Manohar
Asst Prof, IFIM College, Bangalore

1. Introduction

An important study in the field of computer science is the analysis of networks. Internet service providers (ISPs), cell-phone companies, search engines, e-commerce sites, and a variety of other businesses receive, process, store, and transmit gigabytes, terabytes, or even petabytes of data each day. When a user initiates a connection to one of these services, he sends data across a wired or wireless network to a router, modem, server, cell tower, or perhaps some other device that in turn forwards the information to another router, modem, etc. and so forth until it reaches its destination. For any given source, there are often many possible paths along which the data could travel before reaching its intended recipient. When I will initiate a connection to Google from my laptop here at the Sinhgad College, for example, packets of data first travel from my wireless card to my router, then from my router to a hub in the under workings of Sinhgad Server (192.168.60.1), then to the central servers of the Sinhgad College, which transmit them along some path unknown to me until they finally reach one of Google's many servers 40 milliseconds later.

From any one source to an intended destination, however, there are often many different routes that data can take. Under light traffic, the Sinhgad College might distribute the task of sending its Internet users' packets through just two or three servers, whereas under heavy traffic it might use twice or three times as many. By spreading out the traffic across multiple servers, the Sinhgad College ensures that no one server will bog down with overuse and slow down the connection speeds of everyone using it. The idea that there are many possible paths between a source and a destination in a network gives rise to some interesting questions. Specifically, if the connection speed between every two interlinked components is known, is it possible to determine the fastest possible route between the source and destination? Supposing that all of the traffic between two components was to follow a single path, however, how would this affect the performance of the individual components and connections along the route? The optimum solution for the fastest possible transmission of data often involves spreading out traffic to other components, even though one component or connection might be much faster than all the others.

In graph theory, a flow network is a directed graph where each edge has a capacity and each edge receives flow. The amount of flow on an edge cannot exceed the capacity of the edge. Often in Operations Research, a directed graph is called a network, the vertices are called the nodes and edges are called the arcs.

A flow must satisfy the restriction that the amount of flow into a node equals the amount of flow out it, except when it is a source, which has more outgoing flow or sink which has more incoming flow.

In Information technology a network is a series of points or nodes interconnected by communication paths. Networks include the bus, star, token ring and mesh topologies. A network is characterized by the type of data transmission technology in use of TCP/IP. Network can be used to model traffic in a road system fluids in pipes, circuits in an electrical circuit. A network flow is a directed graph where each edge has a capacity and each edge receives a flow.

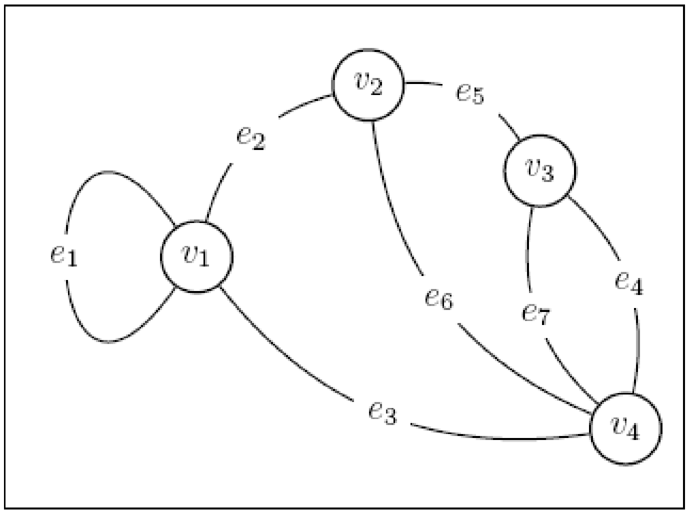
This paper focuses on the maximum rate of flow which is possible from one station to another in the network of telephone lines, highways, railroads, pipelines of (oil or gas or water). This type of network is represented by a weighted connected graph in which the vertices are the stations and edges are lines through which the given commodity (oil. Gas, water, no of messages) flows. Here the weight represents a positive real number which is associated with each edge which represents the capacity of the line that is the maximum amount of flow possible per unit of time.

2. Definition

The study of networks is often abstracted to the study of graph theory, which provides many useful ways of describing and analyzing interconnected components.

To start our discussion of graph theory—and through it, networks—we will first begin with some terminology.

First of all, we define a graph G = (V,E) to be a set of vertices V = {v1, v2, . . . , vm} and a set of edges E = {e1, e2, . . . , en}. An edge is a connection between one or two vertices in a graph; we express an edge ei as an unordered pair (vj , vk), where
vj , vk ∈ V . If j = k, then ei is called a loop.



If we let G denote this graph, then G = (V,E), where V = {v1, v2, v3, v4} and E ={e1, e2, e3, e4, e5, e6, e7}. We can express the edges as e1 = (v1, v1), e2 = (v1, v2), e3 = (v1, v4), e4 = (v2, v4), e5 = (v2, v3), and e6 = (v1, v4). Note that e1 is a loop, since it connects v1 to itself. This type of graph is also known as an undirected graph, since its edges do not have a direction. A directed graph, however, is one in which edges do have direction, and we express an edge e as an ordered pair (v1, v2). Remark that in an undirected graph, we have (v1, v2) = (v2, v1), since edges are unordered pairs.
Sometimes it is convenient to think of the edges of a graph as having weights, or a certain cost associated with moving from one vertex to another along an edge. If the cost of an edge e = (v1, v2) is c, then we write w(e) = w(v1, v2) = c. Graphs whose edges have weights are also known as weighted graphs. We define a path between two vertices v1 and vn to be an ordered tuple of vertices (v1, v2, . . . , vn), where (vj , vj+1) is an edge of the graph for each 1 ≤ j ≤ n − 1. Note that in a directed graph, if the path (v1, v2, . . . , vn) connects the two vertices v1 and vn, it is not necessarily the case that (vn, vn−1, . . . , v1) is a path connecting them as well, since (vi, vi+1) 6= (vi+1, vi). Two vertices v1, v2 are said to be path-connected if there is a path from v1 to v2.
The total cost of a path is the sum of the costs of the edges, so C = Pn−1 j=1 w(vj , vj+1) is the cost of the path from v1 to vn.

3. Algorithms

Our main aim is to find the shortest path and with minimum cost to reach from one source to destination by using the following algorithms.

Dijkstra's Algorithm

It solves the single-source shortest path algorithm for a graph with non negative edges path costs, producing a shortest path tree.This algorithm is often used in routing and as a subroutine in other graph algorithms. For a given source vertex (node) in the graph, the algorithm finds the path with lowest cost (i.e. the shortest path) between that vertex and every other vertex. It can also be used for finding costs of shortest paths from a single vertex to a single destination vertex by stopping the algorithm once the shortest path to the destination vertex has been determined. For example, if the vertices of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road, Dijkstra's algorithm can be used to find the shortest route between one city and all other cities. As a result, the shortest path first is widely used in network routing protocol, most notably IS-IS and OSPF (Open Shortest Path First).

Here are some simple examples of networks:

nodes	**arcs**	**flow**
cities	highways	vehicles
call switching centers	telephone lines	telephone calls
pipe junctions	pipes	water

Algorithm

Let the node at which we are starting be called the **initial node**. Let the **distance of node Y** be the distance from the **initial node** to Y. Dijkstra's algorithm will assign some initial distance values and will try to improve them step by step.

1. Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes.
2. Mark all nodes except the initial node as unvisited. Set the initial node as current. Create a set of the unvisited nodes called the *unvisited set* consisting of all the nodes except the initial node.
3. For the current node, consider all of its unvisited neighbors and calculate their *tentative* distances. For example, if the current node A is marked with a distance of 6, and the edge connecting it with a neighbor B has length 2, then the distance to B (through A) will be 6+2=8. If this distance is less than the previously recorded distance, then overwrite that distance. Even though a neighbor has been examined, it is not marked as *visited* at this time, and it remains in the *unvisited set.*
4. When we are done considering all of the neighbors of the current node, mark it as visited and remove it from the *unvisited set.* A visited node will never be checked again; its distance recorded now is final and minimal.
5. The next *current node* will be the node marked with the lowest (tentative) distance in the *unvisited set.*
6. If the *unvisited set* is empty, then stop. The algorithm has finished. Otherwise, set the unvisited node marked with the smallest tentative distance as the next "current node" and go back to step 3.

4. Network Flows

Imagine that you are a courier service, and you want to deliver some cargo from one city to another .You can deliver them using various flights from cities to cities, but each flight has a limited amount of space that you can use. An important question is, how much of our cargo can be shipped to the destination using the different flights available? To answer this question, we explore what is called a **network flow** graph, and show how we can model different problems using such a graph. A **network flow** graph G=(V,E) is a **directed** graph with two special vertices: the source vertex s, and the sink (destination) vertex t. Each vertex represents a city where we can send or receive cargo. An edge (u,v) in the graph means that there is a flight that flies directly from u to v. Each edge has an associated **capacity**, always finite, representing the amount of space available on this flight. For simplicity, we assume there can only be one edge (u,v) for vertices u and v, but we do allow reverse edges (v,u). Firgure 1 is an example of a network flow graph modelling the problem stated above.

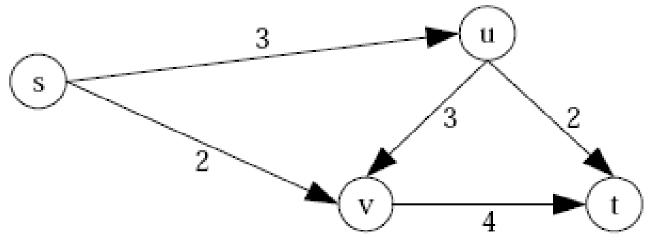


Figure 1. A simple capacitied network flow graph.

With this graph, we now want to know how much cargo we can ship from s to t. Since the cargo
"flows" through the graph from s to t, we call this the **maximum flow problem.** A straightforward solution is to do the following: keep finding
.
paths from s to t where we can send as much flow as possible along each path, and update the flow graph afterwards to account for the used space. The following shows an arbitrary selection of a path on the above graph

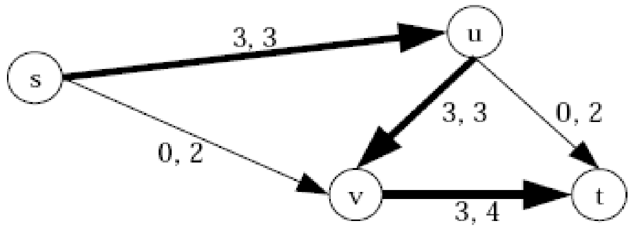


Figure 2. The first number on each edge is the flow, and the second is the capacity.

In figure 2 The first number on each edge is the flow, and the second is the capacity.
In Figure 2, we picked a path s → u → v → t. The capacities along this path are 3, 3, 4 respectively, which means we have a bottleneck capacity of 3 – we can send at most 3 units of flow along this path. Now we send 3 units of flow along this path, and try to update the graph. How should we do this? An obvious choice would be to decrease the capacity of each edge used by 3 – we have used up 3 available spaces along each edge, so the capacity on each edge must decrease by 3. Updating this way, the only other path left from s to t is s → v → t. The edge (s,v) has capacity 2, and the edge (v,t) now has capacity 1, because of a flow of 3 from the last path. Hence, with the same update procedure, we obtain Figure 3 below.

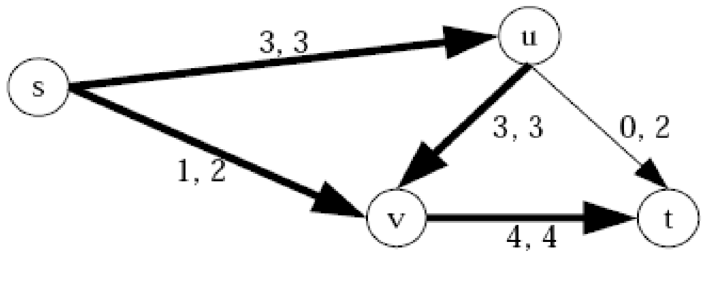


Figure 3. Using path s → v → t. Algorithm ends, but this is not optimal.

Our algorithm now ends, because we cannot find anymore paths from s to t (remember, an edge that has no free capacity cannot be used). However, can we do better? It turns out we can. If we only send 2 units of flow (u,v), and diverge the third unit to (u,t), then we open up a new space in both the edges (u,v) and (v,t). We can now send one more unit of flow on the path s → v → t, increasing our total flow to 5, which is obviously the maximum possible. The optimal solution is the following:

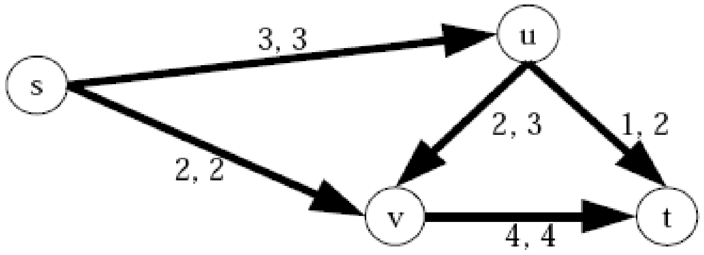
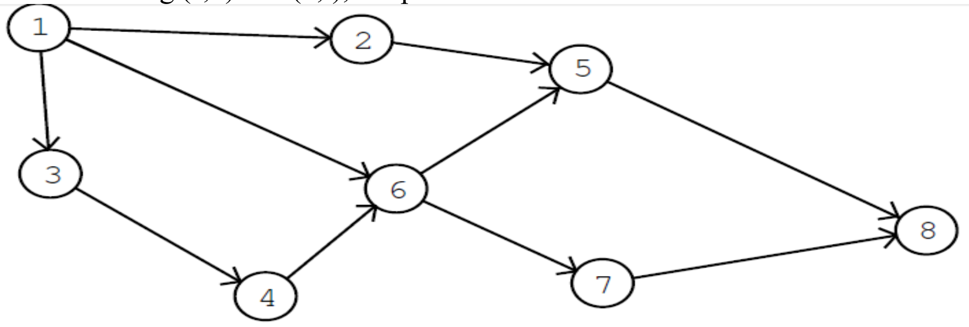


Figure 4. Optimal solution.

So, what is wrong with our algorithm? One problem was that we picked the paths in the wrong order. If we had picked the paths s → u → t first, then pick s → u → v → t, then finally s → v→ t, we will end up with the optimal solution. One solution is to always pick the right ordering or paths; but this can be difficult. Can we resolve this problem without worrying about which paths we pick first and which ones we pick last?
One solution is the following. Comparing Figure 3 and Figure 4, we see that the difference between the two are in the edges (s,v), (u,v) and (u,t). In the optimal solution in Figure 4, (s,v) has one more unit of flow, (u,v) has one less unit, and (u,t) has one more unit. If we just look at these three edges and form a path s → **v** → **u** →t
then we can interpret the path like this: we first try to send some flow
along (s,v), and there are no more edges going away from v that has free capacity. Now, we can **push back** flow along (u,v), telling others that some units of flow that originally came along (u,v) can now be **taken over** by flow coming into v along (s,v). After we push flow back to u, we can look for new paths, and the only edge we can use is (u,t). The three edges have a bottleneck capacity of 1, due to the edge (s,v), and so we push one unit along (s,v) and (u,t), but push **back** one unit on (u,v). Think of pushing flow backwards as using a backward edge that has capacity equal to the flow on that edge. It turns out that this small fix yields a **correct solution** to the maximum flow problem.

5. Networks

A network is characterized by a collection of nodes and directed edges, by called a directed graph. Each edge points from one node to another. Figure below offers a visual representation of a directed graph with nodes labeled 1 through 8. We will denote an edge pointing from a node i to a node j by (i, j). In this notation, the graph of Figure below can be characterized in terms of a set of nodes V = {1, 2, 3, 4, 5, 6, 7, 8} and a set of edges E = {(1, 2), (1, 3), (1, 6), (2, 5), (3, 4), (4, 6), (5, 8), (6, 5), (6, 7), (7, 8)}. Graphs can be used to model many real networked systems. For example, in modelling air travel, each node might represent an airport, and each edge a route taken by some flight. Note that, to solve a specific problem, one often requires more information than the topology captured by a graph. For example, to minimize cost of air travel, one would need to know costs of tickets for various routes.



Directed Graph

5.1 Min-Cost-Flow Problems

Consider a directed graph with a set V of nodes and a set E of edges. In a min-cost-flow problem, each edge (i, j) E is associated with a cost cij and a capacity constraint uij . There is one decision variable fij per edge (i, j) E. Each fij is represents a flow of objects from i to j. The cost of a flow fij is cijfij . Each node j V \ {s, d} satisfies a flow constraint:

$$\sum_{\{k|(j,k)\in E\}} f_{jk} - \sum_{\{i|(i,j)\in E\}} f_{ij} = b_j,$$

where bj denotes an amount of flow generated by node j. Note that if bj is negative, the node consumes flow. The min-cost-flow problem is to find flows that minimize total cost subject to capacity and flow conservation constraints. It can be written as a linear program:

$$\begin{array}{lll} \text{minimize} & \sum_{(i,j)\in E} c_{ij} f_{ij} & \\ \text{subject to} & \sum_{\{k|(j,k)\in E\}} f_{jk} - \sum_{\{i|(i,j)\in E\}} f_{ij} = b_j, & \forall j \in V \\ & 0 \le f_{ij} \le u_{ij}, & \forall (i,j) \in E. \end{array}$$

An important property of the min-cost-flow problem is that basic feasible solutions are integer-valued if capacity constraints and quantitied of flow produced at each node are integer valued.

5.2 Shortest Path Problems

Consider a graph with nodes V and a set of edges E. Think of each node as a city and each edge as a highway that can be used to send a truck from one city to another. Suppose we are given the travel time cij associated with each highway, and we want to get the truck from node o (the origin) to node d (the destination) in minimum time. Suppose that for each (i, j) E, we
Take fij to be a binary variable to which we would assign a value of 1 if edge (i, j) is to be part of the route and 0 otherwise. Then, the route with shortest travel time can be found by solving min-cost-flow problem with bo = 1,bd = −1, bi = 0 for i / {o, d}, uij = 1 for each (i, j) E, and an additional constraint that fij {0, 1} for each (i, j) E.
The additional set of constraints that require flow variables to be binary are introduced because it is not possible to send a fraction of the truck along a highway. With the additional constraints, the problem is not a linear program. Fortunately, we can drop these additional constraints and by doing so obtain a min-cost-flow problem – a linear program for which basic feasible solutions are integer-valued. Because basic feasible solutions are integervalued,if there exists an optimal solution, which is the case if there is a path from node o to node d, then there will be an integer-valued optimal solution. This means a binary-valued solution since each flow is constrained
to be between 0 and 1.

5.3 Max-Flow Problems

Consider a graph with a set of vertices V , a set of edges E, and two distinguished nodes o and d. Each edge has an associated capacity uij but no associated cost. We will assume that there is no edge from d to o. In a max-flow problem, the goal is to maximize the total flow from o to d. In our formulation, nodes do not produce or consume flow, but rather, we introduce an auxiliary edge (d, o) with no capacity limit and aim to maximize flow along this edge. By doing so, we indirectly maximize flow from o to d via the edges in E. We define an augmented set of edges E'= E U {(d, o)}.
The max-flow problem is given by

$$\begin{array}{lll} \text{maximize} & f_{do} & \\ \text{subject to} & \sum_{\{k|(j,k)\in E'\}} f_{jk} - \sum_{\{i|(i,j)\in E'\}} f_{ij} = 0, & \forall j \in V \\ & 0 \le f_{ij} \le u_{ij}, & \forall (i,j) \in E. \end{array}$$

Note that this can be thought of as a special case of the min-cost-flow problem,
where all edges have zero cost, except for (d, o), which is assigned a cost
of cdo = −1.

5.4 Min-Cut Problems

An interesting related problem is the min-cut problem. A cut is a partition of the nodes V into two disjoint sets Vo and Vd such that o Vs, d Vd, and Vo [Vd = V . The objective in the min-cut problem is to find a cut that such that the capacity for flows from Vs to Vd is minimized. One way to represent a choice of partition involves assigning a binary variable to each node. In particular, for each node i V , let pi = 0 if i Vo and pi = 1 if i Vd.Note that for nodes o and d, we always have po = 0 and pd = 1. Further,for each (i, j) E, let qij = 1 if there is an edge directed from i to j, and let qij = 0, otherwise. Note that qij = max(pj −pi, 0). The min-cut problem can be written as

$$\begin{array}{lll} \text{minimize} & \sum_{(i,j)\in E} u_{ij}q_{ij} & \\ \text{subject to} & q_{ij} = \max(p_j - p_i, 0), & \forall (i,j) \in E, \\ & p_d - p_s = 1, & \\ & p_i \in \{0,1\}, & \forall i \in V. \end{array}$$

The decision variables being optimized here include each pi for i V and each qij for (i, j) E. Since the problem involves minimizing a weighted combination of qij ’s, with positive weights, and each qij is constrained to {0, 1}, the constraint that qij = max(pj−pi, 0) can be converted to qij _ pj−pi, without changing the optimal solution. Hence, an equivalent optimization problem is:

$$\begin{array}{lll} \text{minimize} & \sum_{(i,j)\in E} u_{ij}q_{ij} & \\ \text{subject to} & q_{ij} \geq p_j - p_i, & \forall (i,j) \in E, \\ & p_d - p_s = 1, & \\ & q_{ij} \geq 0, & \forall (i,j) \in E, \\ & p_i \in \{0,1\}, & \forall i \in V. \end{array}$$

It is easy to see that the optimal objective value for the min-cut problem is greater than or equal to the optimal objective value for the max-flow problem. This is because for any cut, flow that gets from o to d must pass from Vo to Vd. Hence, the maximal flow is limited by the capacity for flow from Vo to Vd.

The min-cut problem as stated above is not a linear program because the variables are constrained to be binary. As we will now explain, the following linear program also solves the min-cut problem

:

$$\begin{array}{lll} \text{minimize} & \sum_{(i,j)\in E} u_{ij}q_{ij} & \\ \text{subject to} & q_{ij} \geq p_j - p_i, & \forall (i,j) \in E, \\ & p_d - p_s = 1, & \\ & q_{ij} \geq 0, & \forall (i,j) \in E. \end{array}$$

We will not go through the mechanics of converting the max-flow problem to its dual here –that is straightforward to do. However, we will argue that, as a consequence, at each optimal basic feasible solution, each qij is binary valued. At an optimal basic feasible solution, each qij can be viewed as the sensitivity of max-flow to the capacity constraint uij . At an optimal basic feasible solution to the dual, qij is equal to either the rate at which flow increases as a consequence increasing uij or the rate at which flow decreases as a consequence of decreasing uij . Because all capacities are integer-valued, if increasing uij slightly can increase flow, the rate of increase in flow must equal the rate of increase in capacity. The situation with decreasing uij is analogous. Hence, if the sensitivity is nonzero, it is equal to one.

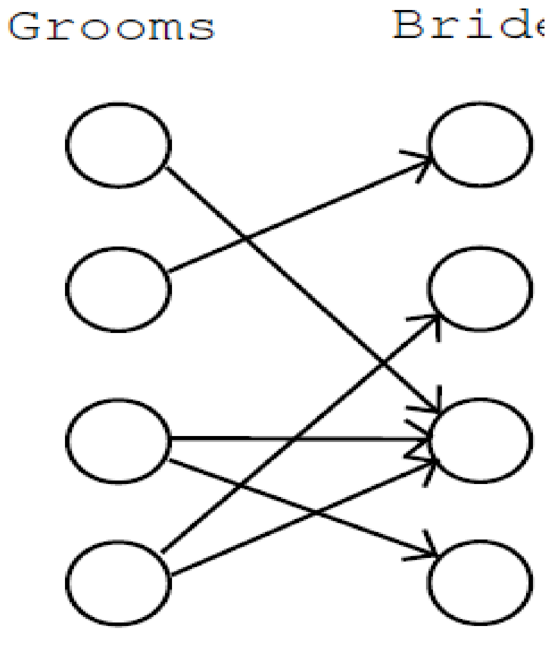
Note that the pis may not be binary-valued; for example, adding a constant to every pi maintains feasibility and does not alter the objective value.So given binary-valued qijs, how do we recover the cut? Well, at an optimal basic feasible solution, qij = 1 if and only if the edge (i, j) goes from Vo to Vd. Based on this, one can easily recover the cut.

5.5 Matching Problems

Consider making matches among n potential grooms and n potential brides.We have data on which couples will or will not be happy together, and the goal is to create as many happy couples as possible. One might represent the compatibilities in terms of a graph as depicted in Figure below :

The problem of finding a matching that maximizes the number of happy couples is known as the matching problem.

Clearly, this problem can be treated as a max-flow problem by adding origin and destination nodes as shown in Figure-



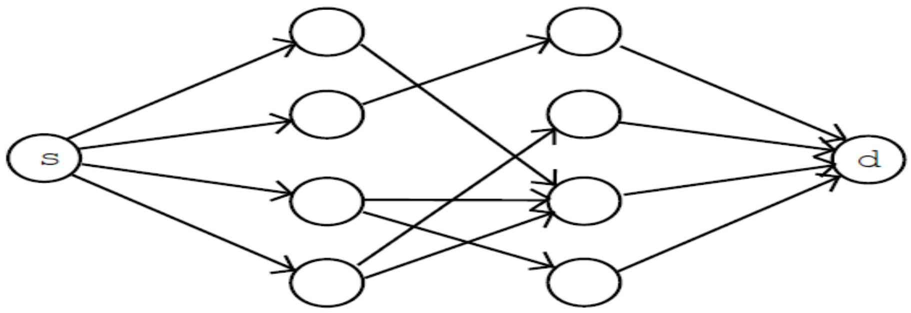
A matching problem.

The capacity of each edge is one. By solving the linear program associated with this max-flow problem, we arrive at a solution that maximizes the number of happy couples. If we are lucky this will turn out to be n. But sometimes, it may not. Interestingly, there is a simple rule for determining whether or not there will be n happy couples without solving the linear program.

5.6 Theorem - (Hall's Marriage Lemma)

There exists a matching such that every bride and every groom is happy if and only if for any set of groom,the set of brides that would form a happy couple with at least one of these grooms is at least as large as the set of grooms.

Proof - It is obvious that for any set A of grooms, there has to be a set of brides of at least equal size in order for there to be any hope of a perfect matching. What is less clear is that there must be a perfect matching if this condition holds for every set of grooms. This follows from the fact that there is a perfect matching if the maximal flow/minimal cut equals n (the number of grooms/brides). It is easy to see that if the conditions on grooms and brides holds, there cannot be any cut of value less than n, and the result follows.



Shortest Path and LP

You are going to visit a national park, and have never been there before. You are using a map to try and make the distance travelled as short as possible.There are 5 intermediate towns, A, B, C, D, and E, you may go through on your way to the park, and the distances between the various locations are given below.

	Miles between Adjacent Towns					
Town	A	B	C	D	E	Destination
Origin	40	60	50	-	-	-
A		10	-	70 -	-	-
B			20	55	40	-
C				-	50	-
D					10	60
E						80

In the table above, a dash means that there is no direct road between the two locations.

Networks and Swimming

The coach of a swim team needs to assign swimmers to a □00-yard medley relay team to compete in a tournament. The problem facing him is that his best swimmers are good in more than one stroke, so it is not clear which swimmer to assign to which stroke. The 5 fastest swimmers and the best times (in seconds) they have achieved with each of the strokes (for 50 yards) are given below.

Stroke	Carl	Chris	David	Tony	Ken
Backstroke	37.7	32.9	33.8	37.0	35.4
Breaststroke	43.4	33.1	42.2	34.7	41.8
Butterfly	33.3	28.5	38.9	30.4	33.6
Freestyle	29.2	26.4	29.6	28.5	31.1

The problem is to try to minimize the sum of the best times for the people competing in the race.

Conclusion

In this paper, we have solved problems in network theory. In network flow we have seen how to deliver some cargo from one city to another using the shortest path.

We have given the definition of network and directed graph and we have listed some problems in Min-Cost Flow, Shortest Path, Min-Cut Problems and Matching Problems.

In matching problem we have tried to match n potential grooms with n potential brides with the help of Hall's Marriage Theorem.

7. References

[1] Cormen, Thomas H. Introduction to Algorithms. 2nd ed. Cambridge, Massachusetts: MIT, 2001.

[2] Dechter, Rina, and Judea Pearl. "Generalized Best-first Search Strategies and the Opti-mality of A*." Journal of the ACM 32.3 (1985): 505-36. ACM Digital Library. Web. 18 May 2010.

[3] Deo, Narsingh. Graph Theory with Applications to Engineering and Computer Science. Englewood Cliffs, NJ: Prentice-Hall, 1974.

[4] Even, Shimon. Graph Algorithms. Computer Science Press, 1979.