Chapter 9

Maximum flows & Maximum Matchings

This chapter analyzes flows and matchings. We will define flows and maximum flows and present an algorithm that solves the maximum flow problem. Then matchings are analyzed in order to prove that flows and matchings are closely related. We will result into the fact that maximum flow algorithms can be used in order to find maximum matchings.

9.1 Flows

9.1.1 Definition of Flows

A network flow graph is can be defined as a graph G = (V, E) and a capacity function c: $V \bullet V \rightarrow R+$ that satisfies all the following conditions:

1. For each $(u, v) \in E$, $c(u, v) > 0$.
2. If $(u, v) \notin E$, then $c(u, v) = 0$.
3. There is a source s and a sink t.
4. Each vertex is on a directed path from s to t.

Due to the fourth condition, in a network flow graph for each node there is a path from source to sink through u therefore the graph is connected.

We can consider that in this kind of networks, flow is "produced" at the source node and "consumed" at the sink node. All the other nodes are just used in order to "move" flow from source to sink and never "store" flow.

Note: Capacity function c is defined independently for the two directions of an edge

A flow in a network flow graph is a function f : $V \times V \rightarrow R+$ such that :

1. For all $(u, v) \in E, f(u, v)$ £ $c(u, v)$ (Capacity constraint : the network flow from one vertex to another must not exceed the given capacity)
2. For all $(u, v) \in V$, $f(u, v) = -f(v, u)$ (Skew symmetry : flow from one vertex u to a vertex v is the negative of the flow in the reverse direction)
3. For all $u \in V - \{s, t\}$, $\sum_{v \in V} f(u,v) = 0$ (Flow conservation : the total flow out of a vertex other than the sink or source is 0)

We should note here that there could be no flow between two nodes that are not connected (there is no edge between them). Flow is only defined across an edge.

The value of the flow passing from source *s* to sink *t* though (al least one) intermediate node v is :

$$|f| = \sum_{v \in V} f(s,v) \;=\; \sum_{v \in V} f(v,t) \qquad (9.1)$$

As we can see the value of the flow is the total outgoing flow of the source or the incoming flow of the sink. Also note that the flow value is constant, meaning that the flow that is "produced" from the source at each moment is equal to the flow that is "consumed" at the sink. This is true, since every other node except source or sink has a net flow value of 0 (due to flow conservation). So the non-zero incoming flow of the sink should match the non-zero outgoing flow of the source.

9.1.2 Maximum flows

A maximum flow in a flow network is a feasible flow (a flow that satisfies the flow conservation) such that the value of the flow is as large as possible.

In the maximum flow problem we are given a flow network *G* with source *s* and sink *t* and we wish to find a flow of maximum value from *s* to *t*. This maximum value depends both on the structure of the network and on the various capacities of its connections. We can consider the problem as finding the maximum quantity of material that we can transfer through a "network" without exceeding the maximum transfer rate of the "network". Assume that we want to provide a city with water, transferring it from a lake through a network of water pipes. Pulping more water that the maximum flow can harm the network while pulping less will not utilize the network at 100%. Also we are not concerned with the time that it takes for the water that is pulped to reach the city, but we should be sure that the amount of water that is pulped per moment is equal to the amount of water that reaches the city per moment.

9.1.3 Network with multiple sources and sinks

We have considered the maximum flow problem as if there is one source leading to one sink. In the case that there are multiple source or sinks, we reduce the problem to the single sink problem by adding a single source at the beginning (supersource) and one at the end (supersink). These nodes are connected to the other, through edges that their capacity is ∞. Now in order to computer the whole flow, we should compute the flow from the supersource to the supersink. Also using edges with capacity ∞, does not affect the value of the total flow.

9.1.4 More definitions

When we think of a maximum flow we can suppose that there exists no path that can "carry" any more flow though the network. If there existed one we could ship more flow through it so the flow is not maximum. So while computing the maximum flow we try to find edges that are not fully utilized and create a path through them. We call residual capacity $c_f(u, v)$ of an edge (*u, v*) on a flow network *G* the additional amount of flow that can be send from *u* to *v* before exceeding the capacity of the edge *c(u, v)*. So

$$c_f(u, v) = c(u, v) - f(u, v) \qquad (9.2)$$

All these edges constitute the residual network: Given a flow *f* on a flow network *G* the residual network G_f of *G* (induced by *f*) is a flow graph on *G* with flows capacities $c_f(u, v)$. The residual network is the actual network that will be used in order to compute the maximum flow. Note that due to the skew symmetry condition ($f(u, v) = -f(v, u)$) an edge that exist in the residual network may not be an edge of the flow network! For each edge of the flow network, there may be created at most one more edge in the residual network. So the maximum number of edges that a residual network may contain is two times the number of edges in the flow network.

We have noted that our purpose is to find more paths from source to sink that provide us with more flow, not utilized until now. These paths are called augmenting paths: Given a graph $G = (V, E)$ and a flow f in G, an augmenting path p is a simple (directed) path connection source s to sink t in the residual graph G_f. Because of the definition of residual network, each edge permits some additional net flow through each edge without violating the capacity constraint. So the maximum amount of flow that we can ship among an augmenting edge is equal to the residual capacity of the edge. The amount of flow that we will ship through the whole augmenting path is the minimum residual capacity of the edges of path. Considering all these, it turn out that if an augmenting path exist in the residual network we can improve the flow of the network. So in order to achieve maximum flow we should search for augmenting paths.

Finally we will define cut of flow networks : An (S, T) - cut in a flow network $G = (V, E)$, is a partition of V to S and $T = V - S$ such that $s \in S$ and $t \in T$.

The flow across a (S, T) - cut is

$$f(S, T) = \sum_{u \in S} \sum_{v \in T} f(u, v) \tag{9.3}$$

The capacity of the cut is

$$c(S, T) = \sum_{u \in S} \sum_{v \in T} c(u, v) \tag{9.4}$$

Cuts in flow networks are very useful. A property of a cut is that the value of the flow across the cuts is equal to the flow of the network. Also this shows that in a flow network, the flow of the network is bounded by the capacity of any cut of the network.

9.1.5 Max flow Min cut Theorem

Theorem 9.1 *Let f be a flow from s to t in G. The following conditions are equivalent*

1) *F is a maximum flow*
2) *The residual graph G_f allows no augmenting paths*
3) $|f|$ *= c(S, T) for some (S, T) – cut*

Proof. (a) (1) → (2) Let F be a maximum flow in a flow network. If the residual graph of the flow network G_f has an augmenting path, the flow f can be improved though that path so flow is not maximum. So there exist no augmenting paths.

(b) (2) → (3) We construct an (S, T) - cut as follows :
Let $S = \{u \in V \mid$ there is a path from s to u in $G_f\}$ and let $T = V - S$. This partition defines an (S, T) – cut : $s \in S$, $t \in T$ otherwise there is an augmenting path in G_f. For each $u \in S$ and $v \in T$ we have $f(u, v) = c(u, v)$ otherwise edge $(u, v) \in E_f$ and $u \in S$. Then $f(S, T) = c(S, T)$ so $|f| = f(S, T) = c(S, T)$

(c) (3) → (1) We have proved that for any (S, T) – cut , $|f| \leq c(S, T)$ thus if $|f| = c(S, T)$ for some cut, f is a maximum flow.

9.1.6 Maximum flows algorithms

The Ford Fulkerson method can be used to solve the maximum flow problem. The method is iterative. It proceeds by finding an augmenting path in the residual network through which we can ship more flow. This is ended when no augmenting path can be found and we have achieved maximum flow. There are many implementations of this method and differs in the way that augmenting paths are discovered.

Ford Fulkerson
Input: a graph G, a source s and a sink t in the graph
Output: a maximum flow for this graph from source to sink

```
Ford Fulkerson ($G, s, t$)
1   for each edge $(u, v) \in$ Edges[$G$]
2   do $f[u, v] \leftarrow 0$
3       $f[u, v] \leftarrow 0$
4   while there exist a path $p$ from $s$ to $t$ in the residual network $G_f$
5      do $c_f \leftarrow \min \{c_f(u, v) : (u, v)$ is in $p\}$
6        for each edge $(u, v)$ in $p$
7          do $f[u, v] \leftarrow f[u, v] + c_f(p)$
8              $f[u, v] \leftarrow f[u, v]$
```

At each iteration of the while loop we find an augmenting path in the residual network and ship more flow through the edges until there exist no more augmenting paths. The running time of the algorithm depends on the implementation. If augmenting paths are found using a BFS search then the algorithm runs in polynomial time. The exact time of the execution in this case is O(VE^2) and is called Edmonds - Karp algorithm.

Example

Given the graph shown in Figure 9.1 (a) we will run the algorithm. The first augmenting path that we find is $a \rightarrow b \rightarrow c \rightarrow d$ (Figure 9.1 (b)). The flow that we can ship through is 4 and so we update the network. In the second iteration (Figure 9.1 (c)) we find $a \rightarrow b \rightarrow d$. Edge (a, b) can carry only 4 additional units of flow so this is the maximum flow through this path. Finally, (Figure 9.1 (d)), we navigate through $a \rightarrow c \rightarrow d$ by adding 2 units of flow since edge (c, d) can't carry any more. Now our network has no augmenting paths so the algorithm terminates and we have found the maximum flow for the flow network

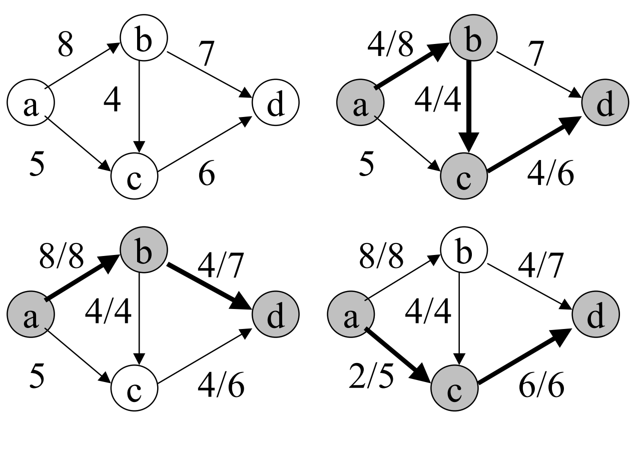


Figure 9.1: A Ford - Fulkerson algorithm example

9.2 Matchings

9.2.1 Definitions

A matching in a graph *G=(V, E),* is a set of edges (links) no two of which have a common end (vertex). In the most practical problems we want to find the matching for the most elements of the graph. Such a matching is called maximum matching.

A matching is also called an independent edge set.

An edge in a matching of a graph is a matched edge**.** All the edges of the graph that are not in the matching are called free edges.

A vertex incident to an edge of the matching is called matched vertex (with respect to the matching) **,** on the other side every other vertex is an exposed vertex (with respect to the matching).

In the most practical problems we want to find the matching for the most elements of the graph, such matching is called maximum matching (also known as maximum cardinality matching).

Formal Definition:
Given a graph *G=(V, E)* a set of edges *M* is a matching if

1. $M \subseteq E$
2. No two edges of *M* share the same node.

A maximum matching in *G* is a matching *M* with maximal cardinality. That means that for every other maximal matching *M'* of the graph *G*: $|M| > |M'|$.

In a bipartite graph $G = (X, Y, E)$, a matching is a complete matching from *X* to *Y* if every vertex in *X* is incident to an edge of the matching. The necessary and sufficient condition for the existence of such a complete matching is that $|f(A)| \geq |A|$ for every subset *A* of *X*, where *f(A)* is the set of vertices that are adjacent to at least one vertex in *A*.

A matching in a graph is a perfect matching if every vertex in the graph is incident to an edge in the matching.

Given an arbitrary graph $G = (V, E)$, we want to find out if all the vertices of V can be grouped in pairs using this matching. According to the Konig's Marriage Theorem: If a bipartite graph $G = (X, Y, E)$ is *k*-regular (where *k* is positive), there is a perfect matching in the graph.

If a graph has a perfect matching *M* then $|M| = |V|/2$, that means practically that each vertex has its unique pair.

Theorem 9.2 (Tutte)
The graph G = (V, E) has a perfect matching if and only if the number of odd components of (G – S) does not exceed |S| for every S Ì V.

9.2.2 Building a Matching

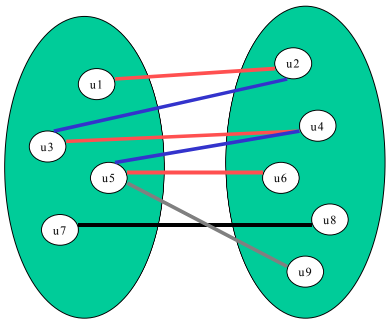
Before continuing with methods and algorithms for finding the maximum matching, we must introduce some definitions of terms.

Given a matching *M* in *G*, a simple path *P* of *G*:

$$P = u_1, u_2, u_3, u_4, \ldots, u_k$$

is the path that connects u_1 with u_k .

An alternating path with respect to *M* is the path *P* that all its odd edges are not in *M* and also all its even edges are in *M* (Figure 9.2).



P = u1 , u2 ,u3 ,u4, u5, u6

M = (u2,u3) , (u4,u5), (u7,u8)

Figure 9.2 : Paths & alternating paths

An alternating path *P* is an augmenting path with respect to *M* if its first and its last node are not covered by *M*.

An augmenting path has odd number of edges. The idea of the augmenting path is one of the fundamental ideas of matching theory. A path or a circuit *P* in a graph *G* is said to be alternating with respect to a matching *M* if its edges alternately are in and not in *M* (an alternating circuit will have an even number of edges). If the end nodes (first and last) of a path *P* are not saturated by *M* then *P* is called an augmenting path.

Removing the even edges of *P* from *M* and adding the odd edges of *P* to *M* increases the size of the matching by one (covering two more vertices).

Theorem 9.3 (Berge 1957)
A matching M in a graph G is maximum iff there is no augmenting path in G with respect to M.

Proof.
(a) Augmenting path ⇒ not maximum: If there is an augmenting path then *M* is not the maximum matching since we can use the path to get a larger matching.

(b) Not maximum ⇒ augmenting path. Assume that *M* is not the maximum. we need to show that there is an augmenting path with respect to *M*. Let *M¢* be a matching in *G* such that |*M¢*| > |*M¢*| consider the graph

H = (V, M È M¢)

The degrees in this graph are 0, 1, and 2. The connected components in the graph are either paths or cycles (circuits), since there are no cycles with odd number of edges, there must be at least one connected component (path) with mode one edge of *M¢* than *M*. This component is an augmenting path.

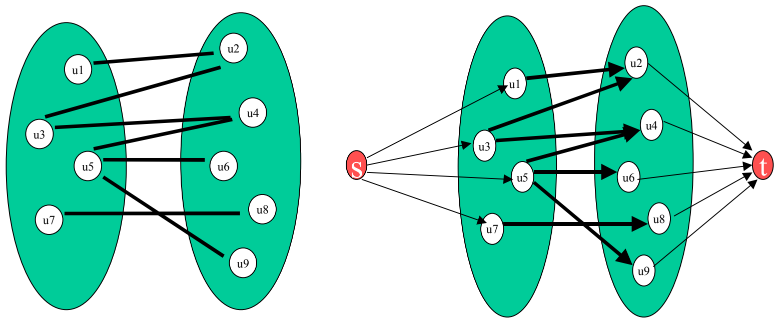
9.2.3 A generalized Matching Algorithm

Main steps:

```
1. $M \leftarrow 0$;
2. while there are augmenting paths with respect to $M$
3.     extend the matching using an augmenting path.
```

The analysis of this algorithm is very complex. Here we will emphasize in a solution for bi-partite graphs only.

In a bipartite graph the search for an augmenting path, which is the most difficult step in the generalized algorithm above, can be done by a variation of the Ford - Fulkenson method which we saw in the Maximum flows.



(a) (b)

Figure 9.3 : Conversion of an undirected bi-partite graph in a flow network in order to find matchings

Given an undirected bi-partite graph *G=(V, E)*, (Figure 9.3 (a)) we must construct a flow network in which the flows will correspond to the matchings.

Let *G¢* = *(V¢, E¢)* be the corresponding flow network for *G* (Figure 9.3 (b))

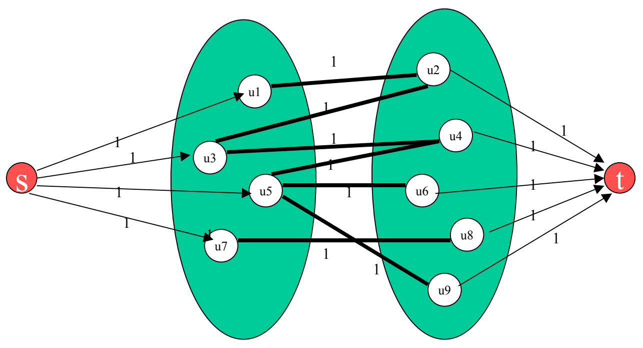
(according to super-source and super-sink theory section 9.1.3) We define two new vertices a source *s* and a sink *t* not in *V*, and we let:

V¢ = VÈ{s, t}

For the original graph *G* if *V* = *L* È *R* where *L* is the left partition, and *R* is the right partition of the bi-partite graph *G*, then the directed edges of *G¢* are given by:

E¢ = {(s, u): u Î L} È {(u, v): u Î L, v Î R, and (u, v) Î E} È {(v, t) : v Î R, }

The last step to finish the directed weighted graph *G¢* is to assign unit capacity to every edge in *E¢*. (Figure 9.4)



P = u1 , u2 ,u3 ,u4, u5, u6

M = (u2,u3) , (u4,u5), (u7,u8)

Figure 9.4 : Last step of conversion, each edge has capacity 1 and the graph has been transformed to an flow network

The following show that a matching in *G* corresponds to a flow in *G¢* network flow.

We say a flow *f* on a flow network *G = (V, E)* is integer-valued if *f(u, v)* is an integer for all (*u, v*) ∈ *V* x *V* .

Lemma 9.4: Let *G = (V, E)* be a bi-partite graph with vertex partition

V = L È R

and let *G¢ = (V¢, E¢)* be its corresponding flow network. If *M* is a matching in *G*, then there is an integer-valued flow g in *G¢* with value |*f*| = |*M*|. Conversely, if *f* is an integer-valued flow in *G*', then there is a matching *M* in *G* with cardinality |*M*| = |*f*|.

Proof. We must show that a matching *M* in *G* corresponds to an integer flow in *G¢*.
If (*u, v*) ∈ *M*, then

f(s, u) = f(u, v) = f(v, t) = 1 and *f(u, s) = f(v, u) = f(t, v) = -1*

For all other edges of *E¢* we define *f(u, v)* = 0.
Since the graph is bi-partite we observe that for each path from *s* to *t*:

s ® u ® v ® t with *u* ∈ *L* , *v* ∈ *R* the capacity is 1.

Since each vertex in *L* has incoming capacity 1 there can be at most 1 edge (*u, v*) leaving *L* to *R*. The set of edges (*u,v*) therefore corresponds a matching *M*.
To see that |*M*| = |*f*| where |*f*| the number of flows, we observe that for every matched vertex *u* ∈ *L* we have *f(s, u)* = 1 and for every edge (*u, v*) ∈ *E* - *M* ,we have *f(u, v)* = 0. Consequently we obtain :

|*M*| = *f(L ,R)*
= *f(L, V¢) - f(L, L) – f(L, s) – f(L, t)*
= *0 – 0 + f(s, L) - 0*
= *f(s , V¢)*
= |*f*|

The above shows that we can use an algorithm for finding maximum flows to find the maximum matching of a Graph *G*.

Bibliography

[1] Thomas H Cormen, Charles E. Leiserson, Ronald L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge Massachusetts, 1990

[2] R. L. Graham, M. Grotchel L.Lovasz. *Handbook of Combinatorics* Vol 1. MIT Press, Cambridge Massachusetts, 1995

[3] V. K. Balakrishnan. *Graph Theory.* McGraw-Hill , New York 1997

[4] R. Diestel. *Graph Theory.* Springer, New York 1997