Matchings

Saad Mneimneh

1 Stable matching

Consider n men and n women. A matching is a one to one correspondence between the men and the women. In finding a matching, however, we would like to respect the preferences of men and women as much as possible. To that end, every individual expresses his/her preference as a total order over the members of the opposite sex. A matching is unstable if there exists two individuals X (man) and y (woman) who favor one another to their current mates. For instance, X is matched to x and Y is matched to y, but X favors y over x and y favors X over Y. X and y represent a dissatisfied pair and will have the tendency to leave their current mates. A matching is stable if no such dissatisfied pair X-y exists. Here's an example:

	1	2	3	4
A	a	b	c	d
B	b	a	c	d
C	a	d	c	b
D	d	c	a	b

	1	2	3	4
a	A	B	C	D
b	D	C	B	A
c	A	B	C	D
d	C	D	A	B

Consider the matching A-a, B-b, C-c, D-d. Then C-d is a dissatisfied pair (C favors d and d favors C). If we change the matching to A-a, B-b, C-d, D-c, then it becomes stable.

2 The proposal algorithm

We repeat the following until all men and women are matched:

1. An unmatched man proposes to the most favorable woman on his list who has not previously rejected him.
2. A woman accepts a proposal from X if she is unmatched, or if her current mate Y is less favorable than X, in which case Y is rejected and reverts to the unmatched state.

Here's an example:

	1	2	3	4
A	b	c	a	d
B	a	b	d	c
C	c	d	a	b
D	b	a	c	d

	1	2	3	4
a	D	B	A	C
b	D	A	C	B
c	D	C	A	B
d	D	A	B	C

- A proposes to b. b accepts A.
- B proposes to a. a accepts B.
- C proposes to c. c accepts C.
- D proposes to b. b accepts D and rejects A.
- A proposes to c. c rejects A.
- A proposes to a. a rejects A.
- A proposes to d. d accepts A.

Theorem: The proposal algorithm always terminates with a stable matching.

Proof: (1) Termination: every step eliminates one woman from a proposer's list. Since the total size of the lists is n^2, the algorithm uses at most n^2 proposals. (2) Matching: Once a woman is matched, she remains so (although she might change to more favorable mates). Therefore, the only way not to end up with a matching is for a woman not to receive a proposal. This means there exists an unmatched man with this choice for a woman on his list that has not been used. The algorithm can proceed. (3) Stability: Let X-x and Y-y be part of the matching and X-y be a dissatisfied pair. Since X favors y, he must have proposed to y before getting matched to x. The only way he could have ended up with x instead is for Y to have rejected his proposal or *dumped* him for another. Therefore, Y must end up with a mate that is more favorable than X (a contradiction).

3 Maximum matching

Another flavor of the stable matching problem is the maximum matching problem. Consider an undirected bipartite graph $G = (V, E)$, i.e. V can be divided into disjoint sets L and R such that $(u, v) \in E \Leftrightarrow u \in L$ and $v \in R$ (an equivalent definition is a graph with no odd cycles). A matching is a set of edges that are node disjoint, and a maximum matching is a matching of maximum cardinality. Given G, construct the directed graph G' by directing every edge from L to R, adding a vertex s with an edge (s, u) for every $u \in L$, and adding a vertex t with an edge (v, t) for every $v \in R$. Finally, assign all edges the capacity 1. It is easy to show that every integral flow f in G' corresponds to a matching

M such that $|f| = |M|$, and vice-versa. Therefore, the value of the maximum flow is equal to the cardinality of the maximum matching. Furthermore, the edges $(u, v) \in E$ with $f(u, v) = 1$ define the matching.

Finding the maximum flow in G' can be done in $O(n^2m)$ time using shortest path augmentation with blocking flows. However, the same algorithm can be proved to run in $O(\sqrt{n}m)$ for this particular class of graphs thus obtained: every vertex, other than s or t, has either a single incoming edge (with capacity 1), or a single outgoing edge (with capacity 1). Such graph is called a unit network. We have the following claims for a unit network:

- there are at most $2\lceil\sqrt{n-2}\rceil$ blocking steps (as opposed to $n-1$)
- a blocking flow can be found in $O(m)$ time (as opposed to $O(nm)$)

To prove the first claim, let f be a blocking flow with residual graph R. Observe that R is also a unit network because f is integral. Therefore, we must have $|f^*| - |f|$ vertex disjoint paths from s to t in R. Since every vertex (other than s or t) is on at most one of the paths, f has an augmenting path of length at most $(n-2)/(|f^*| - |f|) + 1$. This is true for every blocking step. After $\lceil\sqrt{n-2}\rceil$ blocking steps, the shortest augmenting path contains at least $\sqrt{n-2}+1$ edges (because it increases by at least one in every blocking step). Thus $\sqrt{n-2}+1 \leq (n-2)/(|f^*| - |f|) + 1$, i.e. $|f* | - |f| \leq \sqrt{n-2}$. After at most $\lfloor\sqrt{n-2}\rfloor$ additional blocking step, the flow is maximum.

To prove the second claim, observe that augmenting on a path of length k on a unit network saturates at least $(k-1)/2$ edges (either incoming or outgoing). Thus the total time for case 1 in the blocking flow algorithm is $O(m)$. The total time for case 2 in the blocking flow algorithm remains $O(m)$. As a result, a blocking flow can be found in $O(m)$ time.

4 Alternating paths and augmenting paths

Solving for the maximum matching as a flow problem works for the case of bipartite graphs. The flow approach, however, fails if consider the maximum matching problem on arbitrary graphs (no obvious source or sink). In this section, we will develop an augmenting path approach to matchings that will prove to be more suitable for arbitrary graphs. We define a free vertex as one which is not incident to any edge in the matching (unmatched). An alternating path is a path whose edges alternate in and not in the matching. An augmenting path is an alternating path between two free vertices. Note that an augmenting path has exactly one more edge not in M than in M. This notion of an augmenting path is of prime importance due to the following theorem:

Theorem: A matching M is not maximum if and only if there exists an augmenting path p.

Proof: $\Leftarrow$: Let p be an augmenting path. Define $M \oplus p = (M \cup p) - (M \cap p)$. Then $M \oplus p$ is obtained by alternating between adding an edge in p that is not in M, and removing an edge in p that is in M, thus preserving the property of a matching. Thus, $M \oplus p$ is a matching of cardinality $|M| + 1$. $\Rightarrow$: Let M be a matching and M^* a maximum matching, and consider $M \oplus M^*$ (defined the same way above). Hence $M \oplus m^*$ is a matching. In $M \oplus M^*$, every vertex is incident to at most one edge in M and at most one edge in M^*. Therefore, every connected component is either an alternating path, or an alternating cycle. Since $|M^*| > |M|$, there must be a connected component with more edges in M^* than in M. This component can only be an augmenting path (it must start and end with a vertex that is not is not matched in M).

The result suggests a general framework for computing a maximum matching:

Maximum-Matching
$M \leftarrow \emptyset$
while there exists an augmenting path p
  do $M \leftarrow M \oplus p$
return M

On bipartite graphs, the above algorithm runs in $O(nm)$ time since $|M| \leq n$ and an augmenting path (if any) can be found in $O(m)$ time using a slight modification of BFS (explained below) that considers only alternating paths. However, a running time of $O(\sqrt{n}m)$ is possible by augmenting along a maximal set of vertex disjoint shortest augmenting paths simultaneously, in a way analogous to blocking flows using DFS.

The BFS starts with the free vertices in L. Whenever the BFS is in L, it will explore along edges not in M only the vertices in R that have not been seen in a previous visit to R. Whenever the BFS is in R, it will explore along edges in M only the vertices in L that have not been seen in a previous visit to L. When a free vertex in R is among the ones just seen, the BFS stops.

DFS is then performed backwards (why?) on the BFS graph just created to repeatedly find a shortest augmenting path (from a free vertex in R to a free vertex in L), deleting vertices that have been used on such a path (thus obtaining vertex disjoint paths). The whole algorithm can be implemented in $O(m)$ with proper bookkeeping.

Consider the following example with the initial matching given by C-d, D-e, F-c:

	a	b	c	d	e
A			1		
B				1	
C			1	**1**	
D	1	1		1	**1**
E			1	1	
F	1	1	**1**	1	1

Figure 1 shows the graphs produced by the modified BFS. The final (maximum) matching is given by A-c, C-d, D-e, F-a.

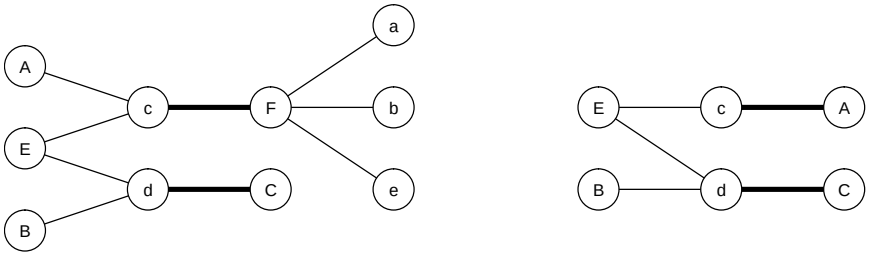


Figure 1: Modified BFS for finding augmenting paths. Thick edges represent edges in the matching. The first time an augmenting path (a, F, c, A) is found by DFS starting from a. The second time, no augmenting path is found, indicating that the matching is maximum.

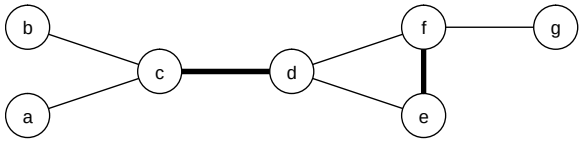
In addition, we have the following two facts:

- the length of the shortest augmenting path increases after every augmentation (stated without proof).
- if a shortest augmenting path for M has length l, the maximum matching M^* satisfies $|M^*| \leq |M| + n/(l+1)$. Proof: there must be at least $|M^*| - |M|$ vertex disjoint augmenting paths for M (consider $M \oplus M^*$). Therefore, if all have length $\geq l$, then we must have at least $(l+1)(|M^*| - |M|)$ vertices. Therefore, $(l+1)(|M^*| - |M|) \leq n$.

Therefore, after $\lceil\sqrt{n}\rceil - 1$ augmentations, $l \geq 2\lceil\sqrt{n}\rceil - 1$. $|M^*| \leq |M| + n/(2\lceil\sqrt{n}\rceil)$. Since $|M|$ increases by at least 1 with every augmentation, $|M|$ will be maximum after at most $\lfloor\sqrt{n}/2\rfloor$ additional augmentations.

5 Maximum matching in general graphs

The modified BFS used above relies on the fact that a bipartite graph does not have odd length cycles. In fact, each vertex has a well defined parity, i.e. its distance from a source is either even or odd, and cannot be both. This property ensures that we can safely ignore cycles (and we are because we do not visit a vertex more than once) without excluding any augmenting paths. In a general graph, cycles cannot be ignored and vertices may have to be visited more than once. The following example illustrates this possibility, starting with the initial matching c-d, e-f:



In the above figure, thick edges are edges in the matching and light edges are not. There is an augmenting path (a, c, d, e, f, g); however, starting BFS from vertex a, for instance, will eventually reach vertices e and f on light edges. The question is then: should BFS visit f from e on a thick edge? In a bipartite graph this would have never been an issue since both e and f would have had the same parity and no edge could have existed between them. Now, not visiting f from e will miss the augmenting path. If, on the other hand, we allow multiple visits to a vertex, visiting f from e might mistakingly identify the augmenting path (a, c, d, e, f, d, c, b).

Let's call a vertex even if it is at an even distance from a starting free vertex s. The anomaly described above occurs whenever we have an alternating path p from s to an even vertex v and an edge from v to an even vertex w on p. The odd cycle formed is called a blossom with w its base.

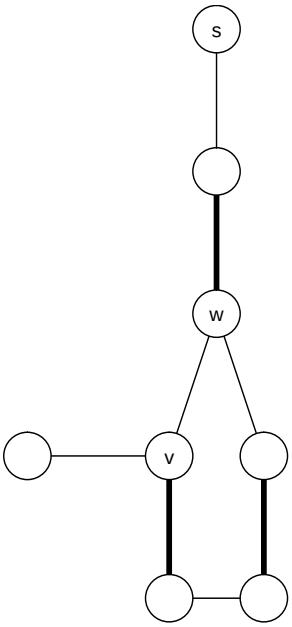


Figure 2: Blossom with base w.

Edmonds was the first to realize that while searching for an augmenting path, a blossom should be shrunk into an even vertex. All vertices that connect to the blossom will connect to this vertex (the shrunken blossom).

Theorem: If G' is obtained from G by shrinking a blossom, then G contains an augmenting path if and only if G' does.

We will not prove this result, but it is easy to see that if G' contains an augmenting path, so does G: if the path in G' does not involve a shrunken blossom, then it is the same path in G. If the path in G' contains a shrunken blossom, then an augmenting path in G can be obtained by expanding the blossom and, starting from its base, going around the cycle either clockwise or counter clockwise (see Figure 2 above, the augmenting path can be obtained by going around the blossom clockwise).

A general strategy for finding an augmenting path is, therefore, to build a search tree starting from a free vertex s. We make s even and all other vertices unreached. We repeatedly grow the tree by adding an edge (u, v) where u is even and shrinking blossoms on the way.

build tree with blossom shrinking
if v is unreached
  then if v is free
        then we found an alternating path
        else (v is matched to some w)
            make v odd and w even
  else if v is even (blossom)
       then shrink the blossom with base the
            least common ancestor of u and v in the tree
            make that shrunken blossom even

Note that an edge (u, v) where u is even and v is odd is ignored in this process. This is because the edge makes a cycle that can be ignored due to the different parities of its vertices (like in the case of bipartite graphs). Once a free vertex v is reached, the augmenting path can be constructed by starting from the root of the tree (or its base it if is a blossom) to the free vertex v. If the path passes through blossoms, they are expanded by going around the cycle from the base to the appropriate exit point, either clockwise or counter clockwise (always even length). Once an augmenting path is found, the search tree can be destroyed and a new one can be built from scratch. We will not discuss the details of the implementation, but ultimately a running time of $O(\sqrt{n}m)$ can be achieved for the entire maximum matching algorithm!