A review of:

An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs

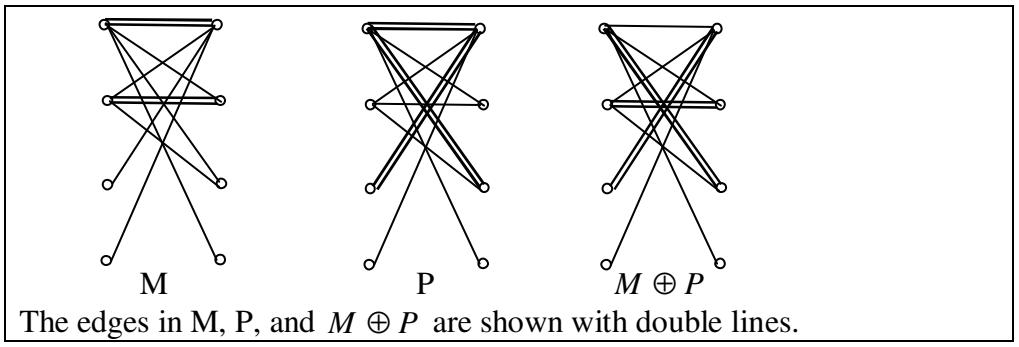
John E. Hopcroft and Richard M. Karp.
review by David Jackson ID# 258894

This paper gives an algorithm to solve the bipartite matching problem with a running time of $O(\sqrt{V}E)$ or $O(V^{5/2})$. The paper contains definitions which the authors hoped would be extended to solve the matching problem in non-bipartite graphs but only includes an algorithm for the bipartite case. A bipartite graph *G=(V,E)* has its set of verticies *V* partitioned into two sets $V = L \cup R$. The edges all go between L and R. A matching $M \subseteq E$ is a set of edges such that at most one edge is incident on each vertex $v \in V$. A vertex $v \in V$ is matched by a matching M if an edge from M is incident upon v.

A simple example of how the algorithm could be applied is the marriage problem. There is a set of girls and a set of boys and the graph contains an edge between every boy-girl pair where they are compatible. The algorithm finds the maximum number of matched couples.

This problem can be solved by using the Ford-Fulkerson maximum flow algorithm as explained in [1]. We run the algorithm on a modified graph: $G' = (V', E')$. $V' = V \cup \{s,t\}$ where s is a source, and t is a sink. Also the set of edges as additional directed edges given by: $E' = E \cup \{(s,u) \mid \forall u \in L\} \cup \{(u,t) \mid \forall u \in T\}$. The cost of running Ford-Fulkerson is O(E|f*|) where |f*| is the maximum flow. In this case E'=O(E) and |f*|=O(V) so the cost of solving the bipartite matching problem is O(VE). The algorithm presented in this paper has a cost of $O(\sqrt{V}E)$ and is the best known algorithm known to date.

This algorithm uses a definition of an augmenting path which is related to the one used for Ford-Fulkerson but different. A simple path P is an augmenting path with respect to M if it starts and ends on an unmatched vertex, and its edges e_i alternate between $e_i \in M - E$ and $e_{i+1} \in M$. For example:



The edges in M, P, and $M \oplus P$ are shown with double lines.

Lemma 1: If M is a matching and P is an augmenting path relative to M, then $M \oplus P$ is a matching, and $|M \oplus P| = |M| + 1$. The set operation $\oplus$ is defined by: $A \oplus B = (A - B) \cup (B - A)$.

Proof: $M \oplus P$ is a matching because it does not contain the edges in $M \cap P$, so each vertex is left with one incident edge. An odd numbered of edges $m = |M \cap P|$ from P are also in M so they are not present in $M \oplus P$. They have been replaced by an even number of edges $n = |P - M \cup P|$ which are in $M \oplus P$. The first and last edges in P are incident on free (non-matched) verticies which are not in M, therefore n = m+1.

The general idea of our algorithm is to use these augmenting paths to increase the number of matchings.

Theorem 1: Let M and N be matchings. If |M| = r, |N| = s, and s > r, then $M \oplus N$ contains at least r - s vertex disjoint augmenting paths relative to M.

This theorem is used to derive the following corollaries which will be useful later on.

Corollary 1: M is a maximum matching if and only if there is no augmenting path relative to M.

Corollary 2: Let M be a matching. If |M| = r, the cardinality of a maximum matching is s, and s > r then there exists an augmenting path of length $\leq 2\lfloor r/(s-r) \rfloor + 1$.

Proof: Let N be a maximum matching, then |N| = s. $M \oplus N$ contains s – r augmenting paths, and each one has at most $\lfloor r/(s-r) \rfloor$ matchings. An augmenting path with this many matchings has length $2\lfloor r/(s-r) \rfloor + 1$ because there are $\lfloor r/(s-r) \rfloor + 1$ odd numbered edges in E - M and $\lfloor r/(s-r) \rfloor$ even numbered edges are in M.

Corollary 2 gives an upper bound on the length of the shortest augmenting path. As the size of the matching increases, the maximum length of the shortest path also increases. This makes sense intuitively because the odd numbered edges of an augmenting path must be in the matching.

Theorem 2: Let M be a matching, P a shortest augmenting path relative to M, and P' and augmenting path relative to $M \oplus P$. Then

$$|P'| \geq |P| + |P \cap P'|$$

This theorem implies that if we were to repeatedly compute the shortest path, each path is either the same length as the previous one or larger, meaning that it increases monotonically. Specifically:

Corollary 3: $|P_{i+1}| \geq |P_i|$

Corollary 4: For all i and j such that $|P_i| = |P_j|$, Pi and Pj are vertex disjoint.
The proof is given in the paper. Because all paths of the same length are vertex disjoint, each one is an augmenting path relative to the same matching M.

Theorem 3: Let s be the cardinality of a maximum matching. The number of distinct integers in the sequence

$$|P_0|, |P_1|, |P_2|, ..., |P_i|, ...$$

is equal to $2\lfloor\sqrt{s}\rfloor + 2$.

Proof: Let M_r be a matching of size $|M_r| = \lfloor s - \sqrt{s} \rfloor$. Using Corollary 2, M_r has an augmenting path P_r such that:

$$|P_r| \le 2\lfloor s - \sqrt{s} \rfloor / (s - \lfloor s - \sqrt{s} \rfloor) + 1 \le 2(\lfloor\sqrt{s}\rfloor - 1) + 1 \le 2\lfloor\sqrt{s}\rfloor + 1$$

For each augmenting path P_i which has been included in M_r, that is, i < r, |Pi| is one of the $\lfloor\sqrt{s}\rfloor + 1$ positive odd integers less than or equal to $2\lfloor\sqrt{s}\rfloor + 1$. The remaining augmenting paths $|P_{r+1}|, ..., |P_s|$ contribute $s - r = \lfloor\sqrt{s}\rfloor$ distinct integers, and the total number of distinct integers is less than or equal to $\lfloor\sqrt{s}\rfloor + 1 + \lfloor\sqrt{s}\rfloor \le 2\lfloor\sqrt{s}\rfloor + 2$.

We can have an algorithm which will run in $2\lfloor\sqrt{s}\rfloor + 2$ phases, where in each phase it computes all of the shortest paths of a particular length. They are all vertex disjoint by corollary 4 and are augmenting paths with respect to the same matching M. The maximum number of matchings, $s = O(V)$, therefore by theorem 3 the number of iterations of the outer while loop is $O(\sqrt{V})$.

Here is the general structure of the algorithm:

Hopcroft-Karp(G)
$M \leftarrow \varnothing$
repeat
    let *l(M)* be the length of the shortest path with respect to M
    let $P \leftarrow \{P_1, P_2, ..., P_k\}$ be a maximum set of vertex-disjoint shortest augmenting paths with respect to M, where all paths have length *l(M)*
    $M \leftarrow M \oplus (P_1 \cup P_2 \cup .... \cup P_k)$
until $P = \varnothing$
return M

If we can find an efficient implementation for finding all of the shortest paths of length *l(M)* inside the repeat-until loop we should be able to find an algorithm which runs faster than the Ford-Fulkerson algorithm. So far, the definitions given here still apply to

general graphs. The remainder of the paper gives an efficient implementation for the bipartite case.

First we define a directed graph, $\overline{G} = (V, \overline{E})$ which the edges in $\overline{E}$ form directed paths. Each edge in E-M runs from R to L and each edge in M runs from *L* to *R*. Then we define a subgraph of $\overline{G}$, $\hat{G}$ where the directed paths in $\hat{G}$ correspond one to one with the augmenting paths in *G*. The graph is divided into levels, where verticies in level Λ_0 connect to the sink t, and s connects to all of the verticies in Λ_{i^*}, where i* is the length of a set of vertex disjoint augmenting paths. The graph has the following properties:

(i) The verticies at even levels $(\Lambda_0, \Lambda_2, ...)$ are in *L* and those at odd levels are in *R*.

(ii) If $(u, v) \in \hat{E}$, then for some I, $u \in L_{i+1}$ and $v \in L_i$

(iii) $\hat{G}$ is acyclic

(iv) The shortest augmenting paths relative to *M* are in one-to-one correspondence with the paths of $\hat{G}$ which begin at a free vertex in *R* and end at a free vertex in *L*. The paths all have length *i**.

The algorithm does a depth first search on this graph for various path lengths *i**. For every vertex u, LIST(u) contains the destination vertex for each outgoing edge from u in arbitrary order. We delete edges from the list as they are explored. B contains the list of verticies which have been visited.

Let TOP = the Top element of STACK

Algorithm **HOPCROFT-KARP**

```
B ← ∅
PUSH s onto STACK
While STACK is not empty do
begin
    While LIST(TOP) is not empty do
    begin
        FIRST=first element from LIST(TOP)
        If FIRST ∉ B then
        begin
            PUSH FIRST onto stack and delete it from LIST
            If TOP ≠ t
                then B ← B ∪ {TOP}
                else
                    begin
                        empty the stack and print the path
                        PUSH s
                    end
        end
        POP STACK
    end
end
```

Proof of correctness:

The loop invariant is as follows:

(i) The stack contains a path from s to TOP

(ii) for every vertex $u \neq s, t$ one of the following cases holds:

(a) $u \in B$ and it is on a path which has been printed

(b) $u \in B$ and it is in the stack on a path which hasn't yet been printed

(c) $u \in B$ and it is not disjoint from any of the paths from s to t which have already been printed

(d) $u \notin B$ and it is disjoint from any of the paths printed so far and u is in LIST(u)

Initialization: When the algorithm starts B is empty, the stack is empty and no paths have been printed.

Maintenance: B contains verticies which have already been visited, and have therefore been matched. It is trivial to show that cases a-d are handled correctly.

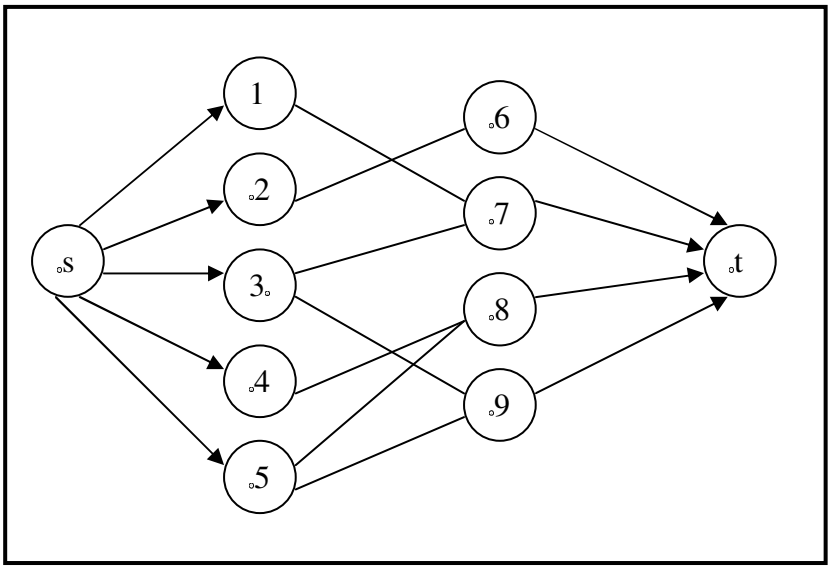
Termination: The algorithm terminates when the stack is empty. At this point, for all u LIST(u) is empty, and B contains all of the verticies.

Complexity

For each iteration of the outer while loop the inner loop builds a set of augmenting paths, it finds a set of vertex disjoint paths, all of the same length. So the outer while loop executes $2\lfloor\sqrt{s}\rfloor+1 = O(\sqrt{V})$ times by theorem 3. In the inner loop each vertex is poped only once and each edge is deleted from each LIST(u) only once, so the inner loop runs in O(V+E) time or O(E) since $E = O(V^2)$. So the total time is $O(\sqrt{V}E)$, or as stated in the title of the paper $O(V^{5/2})$.

Example execution

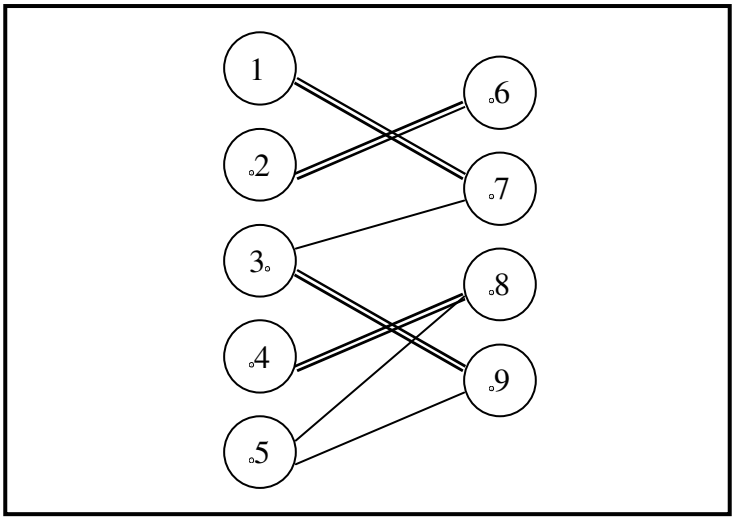
Note that theorem three seems to require that we must find the shortest augmenting path at each step in the algorithm, but this is not the case. Since the adjacency lists are in arbitrary order, the algorithm discovers augmenting paths in arbitrary order. The notion of finding the shortest augmenting path is used only to demonstrate the complexity. The steps of execution are shown for the following input graph.



The stack is shown left to right, bottom to top.

stack	B	other actions
(s)	{}	
(s,1)	{1}	delete 1 from list(s)
(s,1,7)	{1,7}	delete 7 from list(1)
(s,1,7,3)	{1,7,3}	delete 3 from list(7)
(s,1,7,3,9)	{1,7,3,9}	delete 9 from list(3)
(s,1,7,3,9,t)	{1,7,3,9}	Empty the stack and print (1,7) and (3,9)
(s)	{1,7,3,9}	
(s,2)	{1,7,3,9,2}	delete 2 from list(s)
(s,2,6)	{1,7,3,9,2,6}	delete 6 from list(2)
(s,2,6,t)	{1,7,3,9,2,6}	empty the stack and print(2,6)
(s)	{1,7,3,9,2,6}	
(s,3)	{1,7,3,9,2,6}	delete 3 from list(s); list(3) is empty, so pop 3
(s)	{1,7,3,9,2,6}	
(s,4)	{1,7,3,9,2,6,4}	delete 4 from list(s)
(s,4,8)	{1,7,3,9,2,6,4,8}	delete 8 from list(4)
(s,4,8,t)	{1,7,3,9,2,6,4,8}	Empty the stack and print (4,8)
(s)	{1,7,3,9,2,6,4,8}	
(s,5)	{1,7,3,9,2,6,4,8,5}	delete 5 from list(s)
(s,5,8)	{1,7,3,9,2,6,4,8,5}	list(8) is empty, so pop 9
(s,5,9)	{1,7,3,9,2,6,4,8,5}	list(9) is empty, so pop 9
(s,5)	{1,7,3,9,2,6,4,8,5}	list(5) is empty, so pop 5
(s)	{1,7,3,9,2,6,4,8,5}	list(s) is empty, so pop s
()	{1,7,3,9,2,6,4,8,5}	stack is empty; terminates

The resulting matched edges are shown with double lines.



References:

1. John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. SIAM Journal on Computing, 2(4):225-231, 1973.
2. Cormen, Leiserson, Rivest, and Stein. Introduction to Algorithms 2nd ed. Ch 26.3. The Hopcroft-Karp algorithm is described in problem 26-7.