1 Introduction

Back to graphs (and introducing *hypergraphs*). We'll consider problems about *matchings*. As well as being of immense practical significance (in scheduling and task assignment primarily) it was the problem of finding maximum matchings in general graphs that Edmonds was considering when he introduced the Blossom algorithm and speculated about the significance of polynomial-time vs. exponential algorithms.

2 Bipartite graphs and matchings

A *bipartite graph* is a graph, G, whose vertices can be partitioned into two sets A and B such that every edge has one endpoint in A and the other in B. If the graph is connected (meaning that there's a path between any two vertices) then this partition is effectively unique (aside from changing names of the parts). Being bipartite is equivalent to being 2-colourable.

A standard example that motivates the following ideas a little is to think of A as being staff members in a department, and B as a set of papers that need to be taught. A staff member and a paper are adjacent if that staff member can teach that paper.

A *matching*, M, in a bipartite graph (or indeed any graph) is just a set of edges, no two of which share a common endpoint. In the standard example, a matching represents an assignment of (some) staff to (some) papers in such a way that no staff member is responsible for two or more papers, and no paper is taught by more than one staff member.

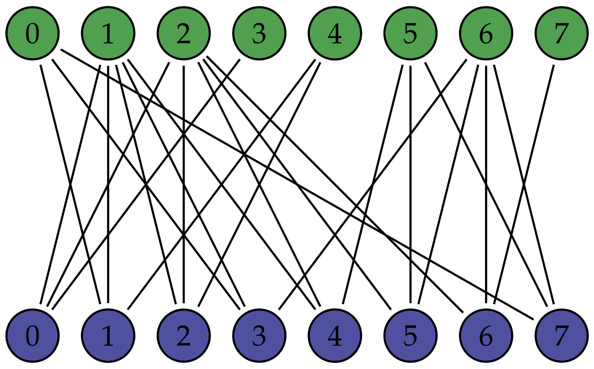
A *maximum matching* in G is a matching whose size is as large as it possibly can be. That is, an assignment of staff to papers that results in the largest possible number of papers being delivered. A matching is *perfect* if every vertex of G is the endpoint of some edge in the matching (every staff member teaches one paper, and every paper is taught by one staff member). Obviously, a necessary condition for a perfect matching to exist in a bipartite graph with parts A and B is that $|A| = |B|$, but this is far from sufficient.

Now we're in a position to ask for various decision and search problems about matchings in bipartite graphs (which we suppose are delivered with a fixed partition into two sets A and B as above)

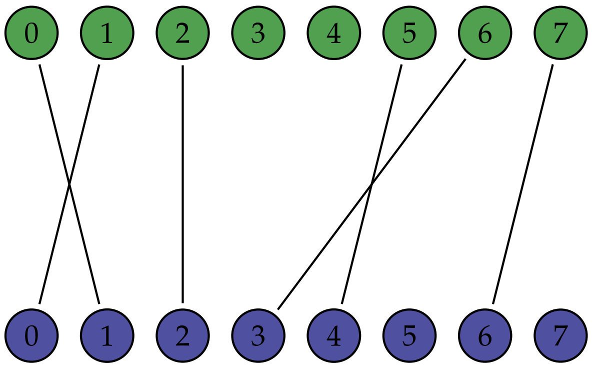
- Does G have a perfect matching?
- Find a perfect matching for G.
- Does G have a matching containing at least k edges? (k a parameter)
- Find a matching in G having at least k edges.
- What is the size of a maximum matching for G?
- Find a maximum matching for G.

All these problems turn out to be in **P**, and using ideas we've seen already it's easy enough to solve them all once you've solved one of them. So, we'll concentrate on the issue of finding a maximum matching.

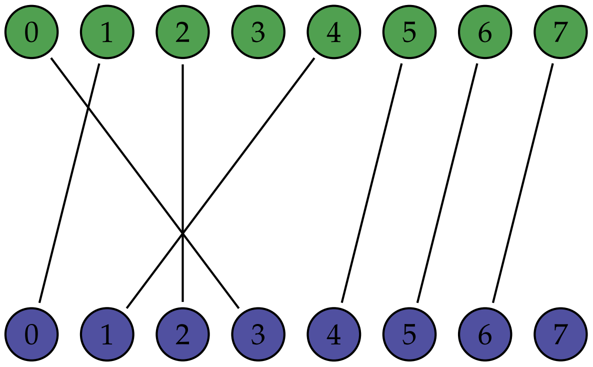
Through the rest of these notes, we're going to use the following graph as an example.



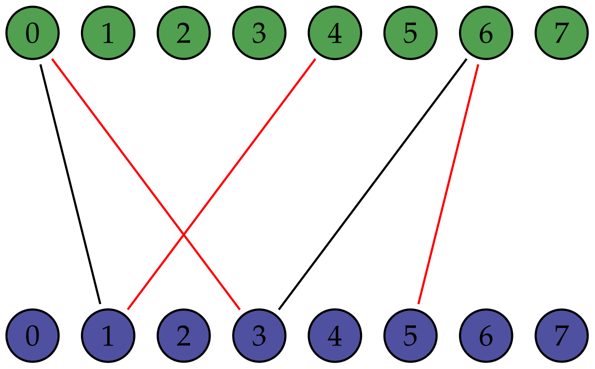
One matching, M_1, in this graph, obtained by choosing the vertices in the bottom part from left to right and then choosing the leftmost adjacent vertex in the upper part that's available to match with is:



A second, larger, matching, M_2 is:



Consider the edges that occur in one or the other of these matchings but not both (their *symmetric difference* – denoted $M_1 \oplus M_2$). The edges of the first matching are coloured black, and of the second red.



Note that these edges form a single path, starting and finishing with a red edge, i.e., an edge that does not belong to the smaller matching. Such a path is called an *augmenting path* for the original matching.

Definition 2.1. Let G be a graph, and M a matching in G. An *augmenting path* for M is a path $v_0 v_1 v_2 \cdots v_n$ of odd length such that v_0 and v_n are not the endpoints of any edge in M and $v_i v_{i+1}$ belongs to M if and only if i is odd. That is, it starts and finishes on unmatched vertices and otherwise alternates between edges in and out of M.

Note that if a matching has an augmenting path it can't be maximum because we could remove all the edges of the path from M and replace them with the other edges of the path – increasing the size of M by one. Of greater interest to us is that the converse is also true:

Proposition 2.2. *If a matching, M, in a bipartite graph, G, is not maximum then it has an augmenting path.*

Proof. Let M' be a maximum matching for G. The symmetric difference $M \oplus M'$ consists of a union of paths and even cycles. The even cycles contain the same number of edges from M as from M'. The paths contain either the same number of edges of each type, or one more from one of the matchings. Since there were more edges in M' to begin with than in M, there are more edges from M' in the symmetric difference than from M and so at least one of those paths must be augmenting for M. □

In fact we can conclude a bit more from the proof – since each augmenting path only accounts for one edge's worth of difference between the sizes of M and M', if there are r more edges in M' than in M then $M \oplus M'$ must contain at least r vertex-disjoint augmenting paths.

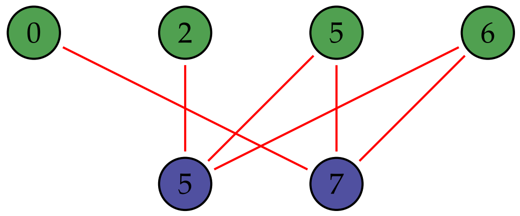
This proposition also gives us the central idea for an algorithm to construct a maximum matching:

```
**Require:** A bipartite graph $G$
**Ensure:** A maximum matching $M$ of $G$
  $M \leftarrow \{\}$
  **while** $M$ has an augmenting path $P$ **do**
    $M \leftarrow M \oplus P$
  **end while**
  **return** $M$
```

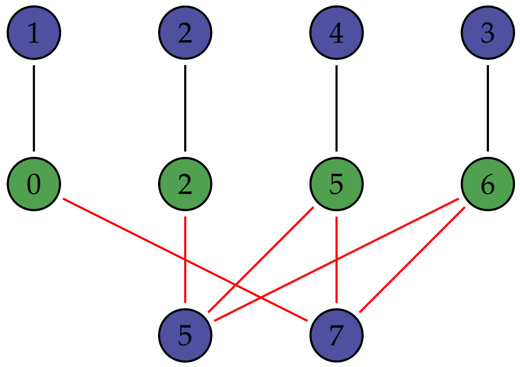
The catch of course is, how do we determine if M has an augmenting path? How do we find one? Or more than one – we could use any set of augmenting paths so long as they have no vertices in common.

3 The Hopcroft-Karp algorithm

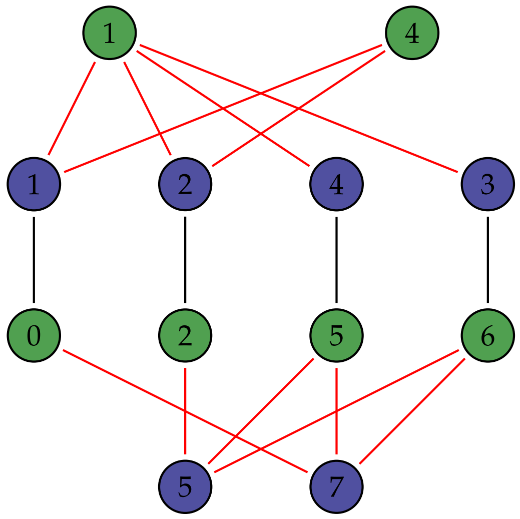
The core of this algorithm is to find a maximal set of shortest vertex-disjoint augmenting paths for a given matching M in a bipartite graph G. This is accomplished by doing a breadth first search forward from the *free*, i.e., unmatched vertices in A alternating between edges that belong to/don't belong to M until we reach at least one unmatched vertex in B. Let's start with illustrating this for our example graph using the initial matching $M_1 = \{01, 10, 22, 36, 45, 67\}$ (shown above). The free (blue) vertices are 5 and 7, which are adjacent to (green) 256 and 056 respectively.



Now (green) 0256 are matched to (blue) 1243 respectively.



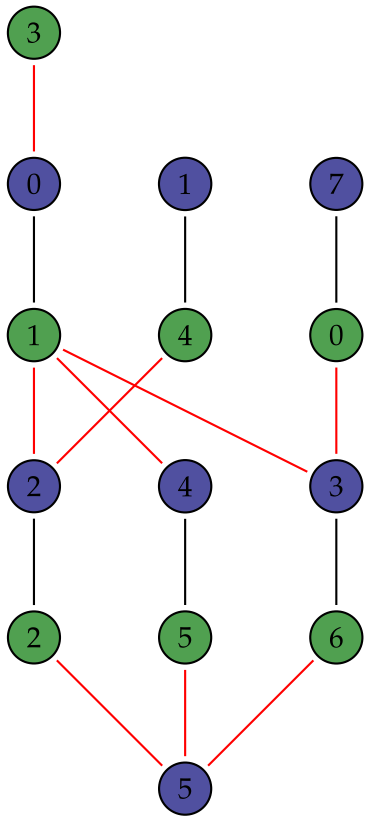
In the next phase we return to using non-matching edges from (blue) 1243 to unvisited green vertices (1347)



At this point we observe that 4 is a free green vertex. So, we have found a shortest augmenting path. We can construct an augmenting path by depth-first search backwards from 4 - say 4-1-0-7. So we take the 10 edge out of the matching and add 70 and 14.

That gives the following matching: 01, 14, 22, 36, 45, 67, 70.

Now let's try again. Start from the only remaining free blue vertex (5) and do a breadth-first search as previously, alternating between edges in or not in the matching. The end product is:



So we find a longer augmenting path between the only remaining pair of free vertices, and hence a perfect matching:

$$03, 14, 21, 36, 45, 52, 67, 70.$$

If we'd reached multiple free vertices on the final layer then, also by depth-first search, we'd try to find augmenting paths back to the original layer – deleting vertices as we use them. This would ensure that we obtain a maximal set of vertex-disjoint shortest augmenting paths.

To summarise, the complete algorithm is as follows:

Algorithm 1 Hopcroft-Karp algorithm for maximum matching

```
Require: A bipartite graph $G$ with parts $A$ and $B$
Ensure: A maximum matching $M$ of $G$
  $M \leftarrow \{\}$
  repeat
    - From the free vertices in $A$ do a breadth-first search alternating edges out of and in
    the matching until you reach a free vertex in $B$ or none are found.
    if one or more free vertices in $B$ are reached then
      - Construct by depth-first search a maximal set of augmenting paths for $M$
      - Update $M$ by switching the augmenting paths
    end if
  until no augmenting path is found
  return $M$
```

Now to complete the analysis of the algorithm we need to do just a little bit more graph theory. The following is a key result from Hopcroft and Karp's original paper. Note that when we talk about an intersection between paths we're thinking of the paths as sets of edges – so the intersection is any edges that they have in common.

Theorem 3.1. *Let M be a matching, P a shortest augmenting path for M, and Q an augmenting path for $M \oplus P$. Then $|Q| \geqslant |P| + |P \cap Q|$.*

Proof. The matching $N = M \oplus P \oplus Q$ has 2 more edges than M, so contains at least 2 vertex-disjoint augmenting paths, P_1 and P_2, relative to M. Since these are augmenting paths for M and P is a shortest such path the length of each is at least $|P|$. So,

$$|P \oplus Q| = |M \oplus N| \geqslant 2|P|.$$

But in general:

$$|P \oplus Q| = |P| + |Q| - |P \cap Q|$$

and so

$$|P| + |Q| - |P \cap Q| \geqslant 2|P|$$

which is the same as

$$|Q| \geqslant |P| + |P \cap Q|$$

□

Starting from a matching M for G we find a maximal set of vertex-disjoint shortest augmenting paths $P_1, P_2, \ldots, P_k$. We then have a new matching $N = M \oplus P_1 \oplus \cdots \oplus P_k$. Now suppose we find an augmenting path Q for N, e.g., in the next round of the algorithm. We first note that Q can't be vertex-disjoint from all of the P's. If it were, and shorter than them, then it would have been a shorter augmenting path for M, while if it were the same length or longer it would contradict the maximality of the chosen P's. Without loss of generality, order the P's so that P_k and Q are not vertex-disjoint. But now notice that P_k is an augmenting path for $M \oplus P_1 \oplus \cdots \oplus P_{k-1}$ and Q is an augmenting path for $M \oplus P_1 \oplus \cdots \oplus P_k$. But, since Q and P_k share a vertex, v, the edge of P_k that belongs to the matching $M \oplus P_1 \oplus \cdots \oplus P_k$ and contains v must also belong to Q. So by the previous theorem:

$$|Q| \geqslant |P_k| + 1.$$

That is, the shortest augmenting path in the next round of the algorithm must always be strictly longer than the shortest path in the preceding round.

Now let V be the set of vertices of G and suppose that we've run the algorithm for $\sqrt{|V|}$-many rounds producing a matching M. This guarantees that the shortest remaining augmenting paths must have length greater than $\sqrt{|V|}$. Suppose that O is an optimal (i.e., maximum) matching, having t more edges than M. By the fact following Proposition **??**, $O \oplus M$ must contain t disjoint augmenting paths relative to M. But, all of them have length at least $\sqrt{|V|}$ and we only have $|V|$ many vertices to work with, so there can be at most $\sqrt{|V|}$ of them, i.e., $t \leqslant \sqrt{|V|}$. Since each round finds at least one new augmenting path if possible, the algorithm can run at most $\sqrt{|V|}$ additional rounds (in fact a little less).

Each round consists of one breadth-first and one depth-first search inside G and so (with just a little care with the data structure) can be carried out in $O(|E|)$ steps, where E is the set of edges of G. Therefore, we can find a maximum matching in time $O(|E|\sqrt{|V|})$.

Therefore, we can find a maximum matching in a bipartite graph in polynomial time, and all the other questions we asked are easily resolved when we can do that.

The best of the YouTube videos I found is this one by Joromy Bou Khalil and Wesley Williams, University of Bristol.

[1] John E. Hopcroft and Richard M. Karp, *An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs*, SIAM Journal on Computing **2** (1973), no. 4, 225-7.

4 Three-dimensional matching

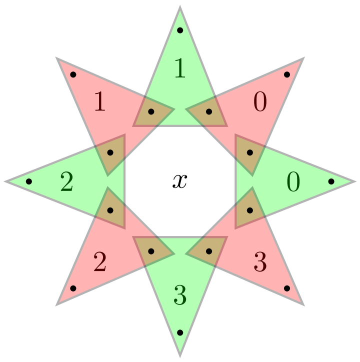
What if time is also a feature when our lecturers teach? For example, what if I'd be willing to teach COSC201 at 1100 but not at 1400? Whereas Anthony might be willing to teach it at 1400 but not at 0900. And so on ...

This is just the matching problem for so called 3-regular hypergraphs. In a 3-regular hypergraph an edge is not a pair of vertices but a triple. That suggests the following problem as a natural generalisation of finding a perfect matching.

PERFECT 3-D MATCHING
Instance A 3-regular hypergraph H.
Problem Does H have a disjoint set of edges covering every vertex exactly once.

Strictly speaking our hypergraph should consist of three sets A, B and C and each edge should contain one vertex from each set. I'll leave you to check that the instance we construct for a reduction of 3-SAT has this property.

Let an instance of 3-SAT be given having k variables and c clauses. For each variable, x, construct a $2c$-pointed star that looks like this if $c = 4$:



The triangles denote edges and an important feature is that the only triangles containing vertices on the inner cycle are the ones shown. Therefore, if a perfect matching is to exist then just on this set it must use either all the red triangles or all the green ones. Suppose it uses the red triangles – this will be associated to setting $x = \mathrm{t}$ because we have "green tips" that still need to covered. The labels on the triangles represent the clause that each one is associated with.

Now for each clause C_i we add two fresh vertices and three triangles. Each of the three triangles contains the two fresh vertices together with the appropriately coloured and labelled tip of the star associated to the corresponding literal. These are the only triangles that contain those two fresh vertices so, if we are to have a perfect matching in the hypergraph then at least one of the literals associated to the clause must be true.

Initially, there are k uncovered tips corresponding to each clause (one per variable) and we only cover one of them with the triangle corresponding to that clause. So, there remain $c(k-1)$ uncovered tips altogether. We clean these up with $c(k-1)$ more "fresh pairs" of vertices and triangles consisting of each such fresh pair and every tip of every star.

Suppose we have a satisfying assignment. Then, we cover each star appropriately, and for each clause select a free tip to cover the vertices associated with the clause. Finally we clean up arbitrarily covering the remaining $2c$ tips with the clean up triangles. That's a perfect 3-D matching.

Conversely, if we have a perfect 3-D matching, the covering triangles of the stars tell us a truth assignment, and the fact that the clause triangles are covered shows that the truth assignment satisfies each clause.

5 Problems

1. Check that all the problems in the list about bipartite matching are polynomially equivalent to one another (basically this just amounts to showing that if you had an algorithm for a more restricted version it could be used to solve a more general version).

2. Check that we can divide the vertices of the hypergraph constructed in reducing 3-SAT to PERFECT 3-D MATCHING into three sets A, B and C such that each triangle contains one vertex from each set.