CS1570 Lecture Notes

Claire Mathieu

Fall 2026

1 October 22: Bipartite matching and the iterated improvement technique

1.1 The iterated improvement technique

In this lecture we introduce an algorithmic technique, *iterated improvement*, and illustrate it on the problem of *bipartite matching*. We will see a basic algorithm for bipartite matching; an improved algorithm (Hopcroft–Karp) is built on the same technique. These notes cover the basic algorithm.

The technique is a method to build a solution S to some problem:

```
S ← empty (or some basic solution)
while possible:
    find a better solution S'
    S ← S'
output S
```

Difference from greedy. In a greedy algorithm, the solution only grows: a choice that has been made is never reconsidered. In iterated improvement, S' may be obtained by *undoing* some of the choices that were made in S. In other words, *you can change your mind*.

1.2 The problem

Definition 1 (Maximum bipartite matching)**. Input:** an unweighted bipartite graph (X, Y, E), with $E \subseteq X \times Y$.

Output: a set of edges $M \subseteq E$ such that every vertex of $X \cup Y$ is incident to at most one edge of M. Such a set M is called a *matching*.

Goal: maximize $|M|$.

Throughout, $n = |X| + |Y|$ is the number of vertices and $m = |E|$ is the number of edges.

Let $X = \{A, B, C\}$, $Y = \{1, 2, 3\}$ and $E = \{A1, A2, B1, B3, C2, C3\}$, where ab denotes the edge between $a \in X$ and $b \in Y$ (this graph is a cycle of length 6; see Figure 1). Some matchings are $\{A1, B3, C2\}$ and $\{A2, B1, C3\}$ (these two have 3 edges, and are the largest possible), $\{A1\}$, $\{A1, C3\}$, and so on.

A greedy algorithm does not work. Consider the following algorithm: start with $M \leftarrow \emptyset$, and repeatedly add an edge to M as long as M remains a matching. On Example 1.2 it could add $B1$, then $C2$, and then it is stuck: the neighbours $1, 2$ of A are taken, and so are the neighbours B, C of 3. The result $\{B1, C2\}$ is not a maximum matching, so the greedy algorithm is *incorrect*.

Other techniques. Divide and conquer, or dynamic programming, do not seem to help: we do not have a good way to define subproblems.

Iterated improvement on the example. Instead, let us use the technique of Section 1.1. We start with $M \leftarrow \emptyset$ and improve M step by step (Figure 1):

- improve: add $B1$; $M \leftarrow \{B1\}$;
- improve: add $C2$; $M \leftarrow \{B1, C2\}$;
- improve: remove $B1$, add $A1$ and $B3$; $M \leftarrow \{A1, B3, C2\}$;
- done.

The third step undoes the choice $B1$ made in the first step.

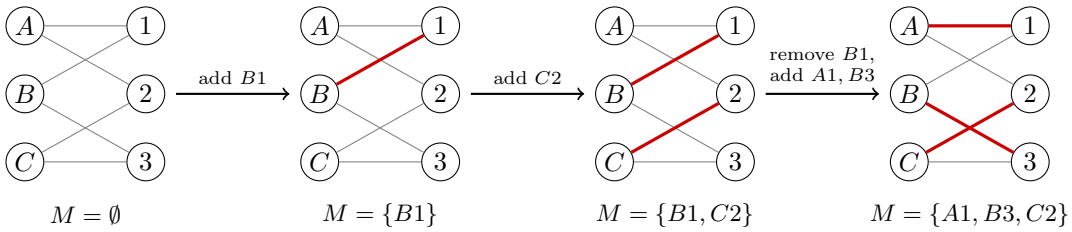


Figure 1: Iterated improvement on Example 1.2. The edges of the current matching are in thick red.

1.3 How to improve a matching

Question. If we have a matching M that is not maximum, how do we improve it?

- One possibility is to add one edge.
- Another is to remove one edge and add two edges.
- $\longrightarrow$?

To see what the general pattern is, look at the edges that are added or removed in the improvements above (and in variants of the example). If we draw only these edges, we observe the following properties of this set of edges. Colour the removed edges red and the added edges purple.

1. It is connected.
2. It is a path (not a cycle).
3. It alternates in colours, i.e. between the edges that are removed (red) and the edges that are added (purple).
4. It has odd length.
5. It starts and ends with purple edges (the edges that you add).
6. It starts and ends at vertices that are not matched in the red matching (the old matching M).

For instance, the third improvement of Figure 1 is the path $A\,1\,B\,3$: the edge $A1$ is added, the edge $1B$ is removed and the edge $B3$ is added; the endpoints A and 3 are not matched by $\{B1, C2\}$.

Answer. Find a path that starts and ends at vertices that are not matched by M, and that alternates between edges in $E \setminus M$ and edges in M.

Definition 2 (Augmenting path)**.** Let M be a matching in (X, Y, E). An *augmenting path for* M is a path (with at least one edge) that starts and ends at vertices that are not matched by M, and that alternates between edges of $E \setminus M$ and edges of M.

Since the two endpoints are unmatched, the first and the last edge of an augmenting path are in $E \setminus M$. So an augmenting path has odd length $2k + 1$, with $k + 1$ edges of $E \setminus M$ and k edges of M. (For $k = 0$ it is a single edge between two unmatched vertices: this is the improvement "add one edge".)

1.4 Augmenting along a path; the algorithm

For a matching M and an augmenting path P (seen as a set of edges), let

$$M \oplus P = (M \setminus P) \cup (P \setminus M)$$

be the symmetric difference: the set of edges that are in exactly one of M and P. We say that we *augment M using P*; see Figure 2.

Claim 1. *If P is an augmenting path for the matching M, then $M \oplus P$ is a matching and has one more edge than M.*

Proof. Let P have $2k+1$ edges, k of them in M and $k+1$ not in M. Then $M \oplus P$ is obtained from M by removing k edges and adding $k+1$ edges, so $|M \oplus P| = |M| + 1$. It is a matching: the vertices outside P are not affected; an inner vertex of P was matched by one edge of $P \cap M$, which is replaced by one edge of $P \setminus M$; and the two endpoints of P were unmatched and are now matched by exactly one edge each. □

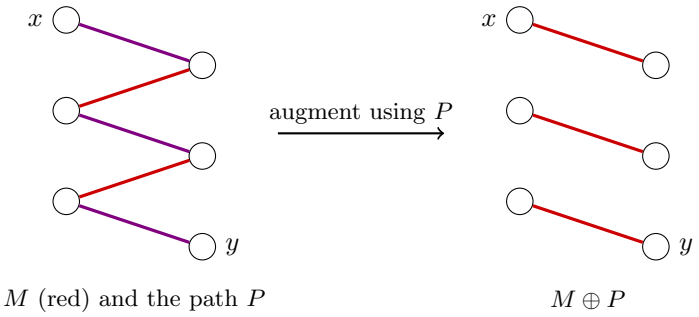


Figure 2: Augmenting along P: the edges of $P \cap M$ (red) leave the matching, the edges of $P \setminus M$ (purple) enter it. The new matching has one more edge.

This gives the following algorithm.

```
M ← ∅
while there is an augmenting path P for M:
    M ← M ⊕ P
output M
```

Theorem 1. *Maximum bipartite matching can be solved in time $O(n \cdot m)$.*

We prove that the algorithm is correct in Section 1.5, and that it can be implemented to run in $O(nm)$ time in Section 1.6.

1.5 Correctness

Fact 1. *If M is a maximum matching then there is no augmenting path for M.*

Proof. If P were an augmenting path for M, then $M \oplus P$ would be a matching with more edges than M (Claim 1). □

Lemma 1. *If M is a matching that is not maximum, then there is an augmenting path P for M.*

This implies the correctness of our algorithm: each iteration increases $|M|$ by one, and $|M| \leq |X|$, so the algorithm stops. It stops only when there is no augmenting path, and then M is a maximum matching by Lemma 1.

Fact 1 and Lemma 1 together say that a matching is maximum if and only if it has no augmenting path (Berge's theorem).

Proof of Lemma 1. Let M be a matching that is not maximum. There is a matching M' that has strictly more edges than M. Consider M and M' (ignoring the rest of the graph).

A first attempt. Since $|M'| > |M|$, there is at least one vertex $x \in X$ that is matched by M' but not by M. Let y_1 be its partner in M' (so $xy_1 \in M'$). The vertex y_1 may or may not be matched in M.

- If y_1 is not matched by M, then xy_1 is an augmenting path.
- Otherwise y_1 is matched in M, by an edge $y_1x_1 \in M$. If x_1 is matched by M' we can continue to follow the alternating path, but if x_1 is not matched by M', then *we are stuck*. We must abandon this attempt and start somewhere else; there must be another possibility for x.

Step back. Consider the graph $H = M \oplus M'$ (on the vertex set $X \cup Y$). It has the following properties.

- It is bipartite.
- Its vertices have degree at most 2: a vertex is incident to at most one edge of M and at most one edge of M'.
- Hence it is a mix of paths and cycles.

Colour the edges of $M \setminus M'$ red and the edges of $M' \setminus M$ purple. In a component of H two consecutive edges share a vertex, so they cannot both belong to the same matching: the colours alternate along every path and cycle. Therefore (see Figure 3):

- the cycles have even length (because H is bipartite) and have as many red edges as purple edges;
- the paths with an even number of edges have as many red edges as purple edges;
- the paths with an odd number of edges have one more red edge than purple edges, or one more purple edge than red edges.

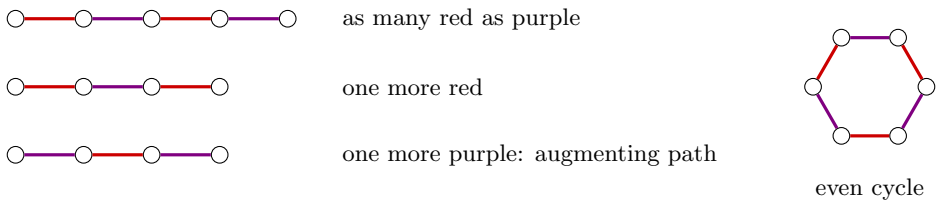


Figure 3: The possible components of $H = M \oplus M'$. Red edges are in $M \setminus M'$, purple edges are in $M' \setminus M$.

Since $|M'| > |M|$ and $|M'| - |M| = |M' \setminus M| - |M \setminus M'|$, the graph H has more purple edges than red edges. If every component had at most as many purple edges as red edges, the same would hold for the sum. So in $H = M \oplus M'$ *there must be a connected component that has more purple edges than red edges.* By the list above, it is a path P of odd length with one more purple edge than red edges, so that it starts and ends with purple edges.

We claim that P is an augmenting path for M. It alternates between edges of $M' \setminus M \subseteq E \setminus M$ (purple) and edges of $M \setminus M' \subseteq M$ (red). It remains to see that its endpoints are not matched by M. Let u be an endpoint of P, and let $e \in M' \setminus M$ be the purple edge of P at u. Suppose that u is matched by M, by an edge $f \in M$. Then $f \neq e$ because $e \notin M$. If $f \notin M'$, then f is a red edge of H at u, so u would have degree 2 in the component P, which contradicts the fact that u is an endpoint. If $f \in M'$, then u would be incident to two different edges e, f of the matching M', which is impossible. So u is not matched by M, and P is an augmenting path. □

1.6 Implementation and running time

We need to implement the step "look for an augmenting path for M": given (X, Y, E) and a matching $M \subseteq E$, find an augmenting path or decide that there is none. For now, imagine that you know where to start: an unmatched vertex $x \in X$. Then this is a *graph exploration* problem, once we *change the representation* of the instance (we do not have the better matching M' of the proof, only M).

The directed graph D_M. Create a directed graph on the vertex set $X \cup Y$ as follows.

- Every edge in $E \setminus M$ is directed from X to Y.
- Every edge in M is directed from Y to X.

Then we can rephrase the problem as: *find a directed path from some unmatched $x \in X$ to some unmatched $y \in Y$*. This is a *reachability* problem. Indeed, a directed path starting in X alternates between edges of $E \setminus M$ (going from X to Y) and edges of M (going from Y to X); it starts and ends with an edge of $E \setminus M$ when it goes from an unmatched $x \in X$ to an unmatched $y \in Y$. Conversely, an augmenting path, directed from its endpoint in X to its endpoint in Y, is a directed path of D_M.

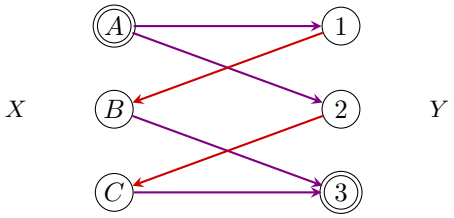


Figure 4: The directed graph D_M for $M = \{B1, C2\}$ in Example 1.2: edges of $E \setminus M$ (purple) go from X to Y, edges of M (red) go from Y to X. The unmatched vertices are drawn with a double circle. The directed path $A \to 1 \to B \to 3$ is the augmenting path found in Figure 1.

Where to start. Instead of guessing the starting vertex x, search from all unmatched vertices of X at the same time. For example, add a new vertex s with an edge to every unmatched $x \in X$, and run a breadth-first or depth-first search from s in D_M. If the search reaches an unmatched $y \in Y$, the path found (from s to y, without s) is an augmenting path.

Proof of Theorem 1. The correctness was shown in Section 1.5. Each iteration of the algorithm increases $|M|$ by one (Claim 1), and $|M| \leq |X| \leq n$, so there are at most n iterations that find a path, and one last iteration that finds no path. In each iteration we build D_M in time $O(n+m)$, run one graph search in time $O(n+m)$ and, if a path is found, augment M along it in time $O(n)$. The total is $O(n(n+m))$. Isolated vertices can be removed beforehand, in time $O(n+m)$; then $m \geq n/2$, so $n+m = O(m)$ and the total is $O(nm)$. □