Graph Algorithms (Spring 2025)

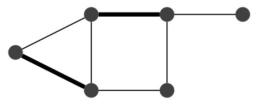
6. Maximum Matchings

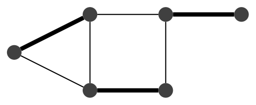
Teacher: Arnaud Casteigts *Assistant: Matteo De Francesco*

In this course, we motivate and introduce the problem of computing matchings in graphs. We define matchings and discuss the difference between maximal and maximum matchings. Then, we present the concept of *augmenting paths* and give a simple algorithm for solving the problem in bipartite graphs. For general graphs, we present the main ideas used in Edmonds' algorithm, namely *alternating trees* and *blossom contractions*. Finally, we briefly discuss the case where the input graph is weighted.

6.1 Matchings

Let $G = (V, E)$ be a graph. A **matching** in G is a subset of edges $M \subseteq E$ that are independent (i.e., they do not touch each other). Here are some examples:





The set $M = \emptyset$ is a valid matching, however it is not very interesting. In general, the goal is to maximize the size of a matching. There are two distinct notions of maximality: the matching is **maximal** if no additional edge can be added to it, and it is **maximum** if no larger solution exists. Obviously, a maximum matching is also maximal, but the converse is not true. In the above examples, the matching on the left is maximal, but not maximum. The matching on the right is maximum.

Maximal matchings can be computed using a simple greedy algorithm:

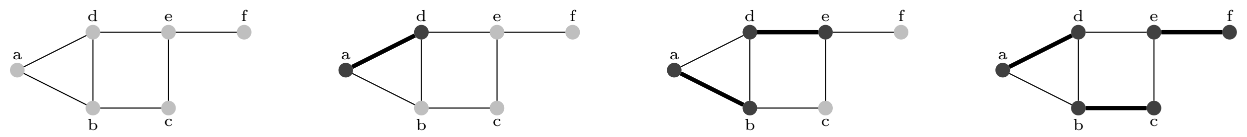
1. Initialize $M := \emptyset$
2. Find an edge that is independent from the edges of M and add it to M.
3. Repeat step 2 until no further edge can be added.

Maximal matchings turn out to be useful in many scenarios. As one could expect, maximum matchings are harder to compute. For a long time, it was open whether this problem is even polynomial time solvable. It turns out that it is, and this is the object of this class. (We shall return on maximal matching in a subsequent class on approximation algorithms.)

6.2 Augmenting paths

The computation of maximum matching relies heavily on the concepts of **exposed vertex** and **augmenting path**. At any step of the computation, a vertex is *exposed* if it does not participate to any edge of the current matching (we also say it is *unmatched*). An *augmenting path* is a path from an exposed vertex to another exposed vertex that alternates between edges in the matching and edges outside the matching. The key property is that if an augmenting path P exists, then one can flip the status of its edges in the current matching M: the edges of P that are in M are removed from M and the ones not in M are added to M (in other words, M becomes $M \triangle P$), which produces a new matching of size $+1$.

Let's do this on an example, using the same graph as before. Initially, M is empty and all the vertices are exposed (light gray color in the picture). At this point, every edge is itself an augmenting path, so we have many choices. Let's choose the edge ad and flip its



status in M (i.e. add ad to M). We can then find another augmenting path, for example $P = (b, a, d, e)$ and flip its status in M, removing ad and adding instead ab and de. Finally, we find the augmenting path $P = (c, b, a, d, e, f)$ and flip these edges again. The resulting matching is now maximum. Observe that, in this example, G no longer has any exposed vertices. Such a matching is called a **perfect matching**. Perfect matchings do not always exist. For example, if the number of vertices is odd, or if the structure of the graph prevents it, like in a star graph (✳) where the maximum size of a matching is 1.

There is a nice theorem by Claude Berge (1951):

Theorem 6.1. *A matching is maximum if and only if there is no augmenting path.*

Proof. We need to prove the two sides of the implications: (1) If M is maximum, then there is no augmenting paths; and (2) If M is not maximum, then there exists an augmenting path.

(1): (Contrapositive) If there exists an augmenting path, then M can be increased, thus it is not maximum.

(2): Let M be a non-maximum matching, then there exists a matching M^* that is larger than M. Let $G' = (V, E') \subseteq G$ with $E' = M \triangle M^*$ (the symmetric difference of M and M^*, i.e. the edges that are in M or M^*, but not in both). First observe that all the vertices in G' have degree at most 2 (since both M and M^* are matchings). This implies that the connected components of G' are either cycles, paths, or isolated vertices. Moreover, the cycles and the paths alternate between edges of M and edges of M^*. Now, observe that G' has more edges from M^* than from M (since $|M^*| > |M|$). Let's unfold this: All the cycles of G' must have even length (why?), and thus as many edges from M and M^*, which implies that at least one path has *more* edges from M^* than from M, thus its two endpoints are exposed in M. This path is an augmenting path. □

This theorem suggests a pretty simple strategy: find augmenting paths repeatedly as long as we can, then the matching is maximum. The main problem is now to find such augmenting paths!

6.3 Maximum matchings in bipartite graphs

The special case of bipartite graphs is interesting because it is simpler to solve, and it has many applications, in particular for assignment problems. Suppose you have a set of employees (one part) and a set of tasks (the other part). There is an edge between an employee and a task if and only if this employee is able to realize this task. Then, a maximum matching describes an optimal assignment of tasks in the company. Other examples include the assignment of classrooms to lectures, taxi cabs to customers, or candidate students to higher education formations (e.g., *Parcoursup*[1] uses a bipartite matching algorithm).

There is a nice trick for finding an augmenting path in a bipartite graph. Let $G = (U, V, E)$ be a bipartite graph where U and V are the two parts. Let's say we already computed a matching M that is not yet maximum, like in Figure 1. The trick is to turn G into a *directed* graph G' as follows: First, add two vertices s (source) and t (target). For

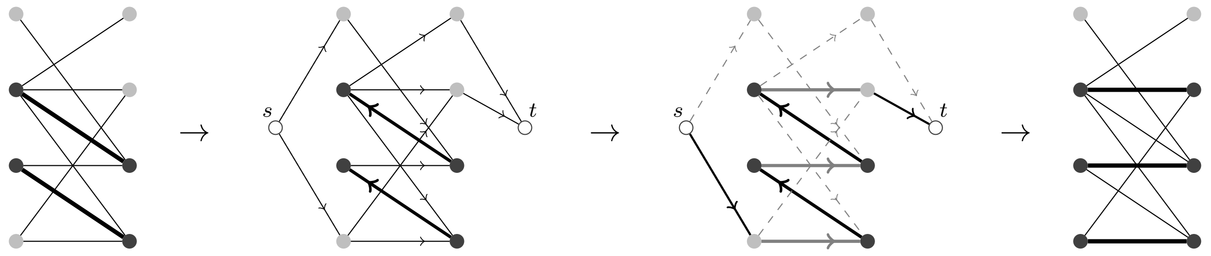


Figure 1: Augmenting paths in bipartite graphs.

every exposed vertex v in U, add an arc from s to v, and for every exposed vertex v in V, add an arc from v to t. Finally, for every edge $uv \in E$ such that $u \in U$ and $v \in V$, turn it into an arc *from u to v*, except for the edges of M, that we orient in the opposite direction. We now have the nice following fact: Every path from s to t in G' defines an augmenting path in G, and vice versa.

How to find a directed path from s to t? That's easy: just run a traversal (DFS or BFS) from s, which costs $O(m)$. Since the matching increases in each iteration and its maximum size is n, we can repeat at most $O(n)$ times this procedure, which gives a total cost of $O(nm)$.

[1]Plateform used by the French higher education system.

6.4 Maximum matchings in general graphs

In 1965, Jack Edmonds published the first polynomial time algorithm for finding a maximum matching in an arbitrary graph. Its algorithm has a time complexity of $O(n^3)$. In this section, we will present the main components of this algorithm.

As before, augmenting paths play a central role here. The heart of the algorithm is to find them efficiently. The main two components are **alternating trees** and **blossoms**.

6.4.1 Alternating trees

For now, let's suppose again that the graph is bipartite and that we have a matching M that may or may not be maximum (see Figure 2). Alternating trees offer a different way to find augmenting paths (referred to as the *Hungarian method*). We start by picking an exposed vertex u (say, u_4 in our example). For each neighbor v of u, we do the following. If v is exposed, we have found an augmenting path. Otherwise, v must be matched with some other node w. We then add at once v, w, uv and vw to the tree. Once all the neighbors of u have been considered, we pick a leaf of the tree as the new node u and repeat.

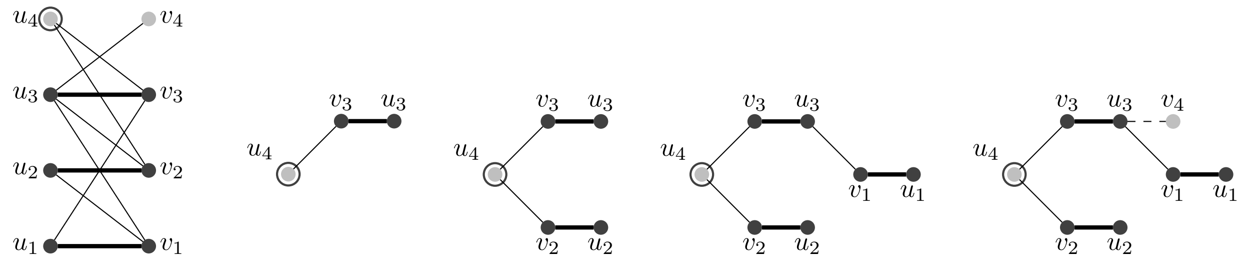


Figure 2: Construction of an alternating tree.

Note that the alternating tree may not be unique, even for a given start exposed vertex. However, the good news is that, if the graph is bipartite and if an augmenting path exists, this procedure will find it (possibly starting at several exposed vertices if the graph has several components). Unfortunately, this technique does not work when the graph is not bipartite, because it fails in presence of *odd cycles*.[2]

6.4.2 Blossoms

If the graph has odd cycles, then the alternating tree may fail to find an augmenting path. This is illustrated by the example on Figure 3. Here, building a tree from the exposed vertex on the left does not reveal any augmenting path. Yet, one exists if we turn anti-clockwise in the cycle. (Interestingly, starting at the other exposed vertex would work here, but ignore this, we could build a more complex example where the tree fails from any exposed vertex.)

[2]Recall that a graph is bipartite if and only if it has no odd cycles.

Let's look more closely at the configuration. Here, we have an alternating path between

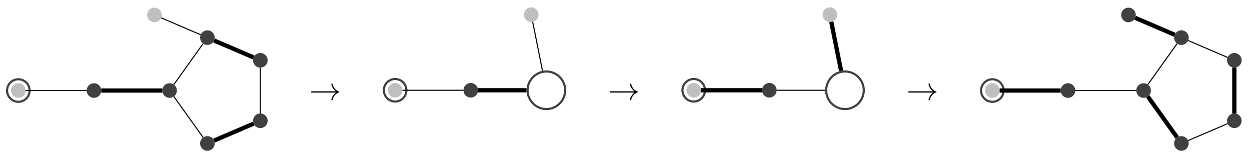


Figure 3: Example of a blossom contraction and expansion.

our start vertex and the cycle, and an alternating cycle that is already saturated (k edges are in the matching, for a total of $2k + 1$ edges in the cycle). In this case, the path is called the **stem** and the cycle is called the **blossom**. Edmonds' main idea is to contract the blossom into a single node, doing so several times if the graph has several blossoms, then run the alternating tree algorithm on the contracted graph. The key property is that, if an augmenting path is found in the contracted graph, then it can *always* be lifted up to the original graph, resulting in a matching of increased size (as in the picture). In fact, we have the following stronger theorem:

Theorem 6.2 (Edmonds, 1965)**.** *For a given matching M, an augmenting path exists in the original graph if and only if it exists in the contracted graph.*

We will not cover the details of the proof. The actual algorithm deals with a number of subcases that are analoguous to the above one. The main conclusion is that a maximum matching can be obtained using this procedure, and so, in time $O(n^3)$. One factor n comes from the number of times the matching can be augmented, starting from an empty matching; another n^2 factor comes from the blossom contractions (and subsequent expansions). The construction of the alternating tree in the contracted graph can essentially be performed by a traversal (DFS or BFS) in time $O(m)$, which is dominated by the $O(n^2)$ cost of contractions, thus this part is "for free".

6.5 How about weighted graphs?

Let $G = (V, E, w)$ be a weighted graph. A maximum matching M in G is one such that $\Sigma_{e \in M} w(e)$ is maximum. In bipartite graphs, one typically uses the *Hungarian method* with weights (possibly given by a matrix). It is also possible to adapt the strategy from Section 6.3 using shortest paths algorithms instead of naive traversals when playing with the directed graph, although I didn't check the details. An important remark is that Edmonds' algorithm actually works for both the weighted and unweighted version, so it could be used in all situations (including bipartite weighted graphs). The weighted version of the algorithm requires playing with the costs dynamically (modifying them during the execution), and has a slightly higher time complexity, namely $O(n^2 m)$. A more complex algorithm was introduced later by Micali and Vazirani, with time complexity $O(m\sqrt{n})$. Finally, the bipartite weighted case can also be solved using Matroid intersection, of which we shall say a word (or perhaps two) in a subsequent class.