Elementary Introduction to Graph Theory

(01YEIG 2026/2027)

Lecture 9

Francesco Dolce, PhD
dolcefra@cvut.cz

November 20, 2026

updated: November 13, 2026

PDF available at the address:
dolcefra.pages.fit/ens/2627/YEIG-lecture-09.pdf

1 Augmenting Semi-Path Search

A systematic implementation of the Ford-Fulkerson Algorithm in a network $N = (D, s, t, c)$ search for an f-augmenting semi-paths using a Tree-Search Algorithm.

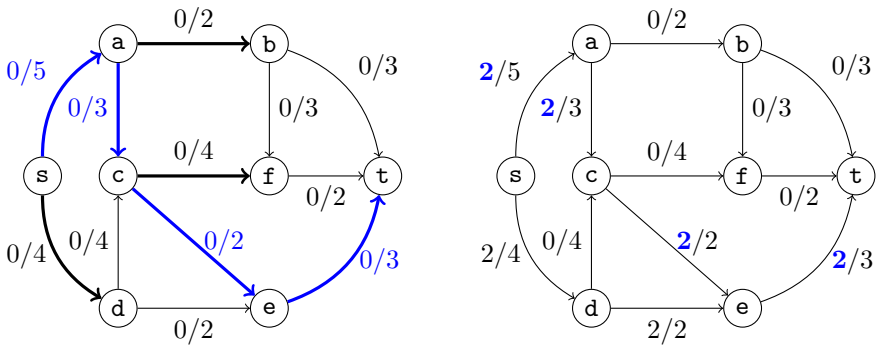
We start, as before, with the zero flow.

At each step, we construct an spanning rooted tree $T(s)$, i.e., a rooted tree with root the source s, such that for every vertex v of the tree, the unique semi-path from s to v is f-admissible. If the tree reaches the sink t, then the unique semi-path in the tree from s to t is an f-augmenting semi-path and we can replace f by the flow $\tilde{f}$ as seen before.

This Tree-Search Algorithm is known as *Augmenting Path Search* (APS).

Example 1 Consider the network N of Figure 1a with zero flow f_0. A possible admissible search tree for f_0 obtained by DFS is shown in Figure 1a on the right (arcs in bold). The semi-path $P_1 = (\mathtt{s}, \mathtt{a}, \mathtt{c}, \mathtt{e}, \mathtt{t})$ is f_0-augmenting with residual capacity 2.

The flow f_1 obtained by augmenting f_0 along P_1 is shown in Figure 1b.



(a) An APS in a network with zero flow.

(b) Network with augmented flow.

Exercise [Augmenting Path Search]. Use the Augmenting Path Search to find a maximum flow in the network in Example 1.

2 Time complexity of Ford-Fulkerson Algorithm

The running time complexity of the algorithm above, called *Ford-Fulkerson Algorithm*, is, for a network with integer capacities $\mathcal{O}(\#E \cdot \mathrm{val}(f))$, where f is a maximum flow.

Indeed, if the algorithm chooses augmenting semi-paths poorly, it can do a lot of unnecessary steps.

Example 2 Consider the network in Figure 2.

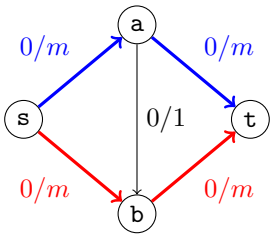


Figure 2: A network with zero flow.

Obviously, the value of a maximum flow is $2m$, which can be easily found in 2 steps: using the augmenting semi-paths $(\mathtt{s}, \mathtt{a}, \mathtt{t})$ and $(\mathtt{s}, \mathtt{b}, \mathtt{t})$.

However, let us assume that the algorithm will alternatively use semi-paths of the forms $(\mathtt{s}, \mathtt{a}, \mathtt{b}, \mathtt{t})$ and $(\mathtt{s}, \mathtt{b}, \mathtt{a}, \mathtt{t})$. The first three steps are shown in Figure 3.

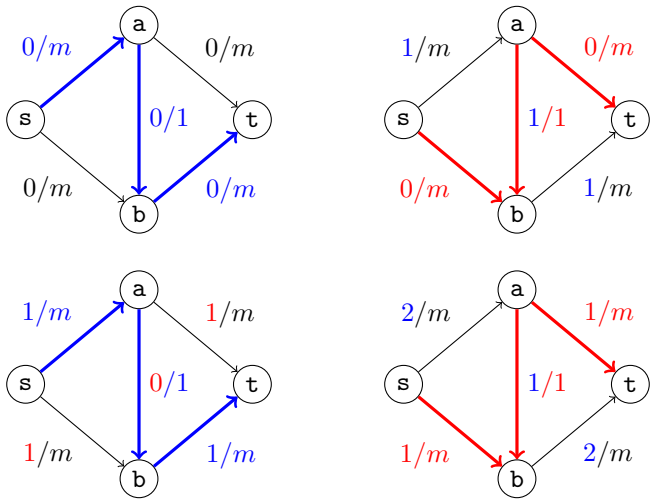


Figure 3: Initial configuration (top-left), first step (top-right), second step (bottom-left) and third step (bottom-right) in the Ford-Fulkerson algorithm.

The algorithm will eventually find the right value of a maximum flow, $2m$, but it will take $2m$ steps.

This is one of many cases showing that Ford-Fulkerson algorithm is sensitive to the choice of augmenting semi-paths.

Edmonds and Karp proved that most of these problems can be avoided if we choose, at each step, an augmenting semi-path with the minimum number of traversal. The time complexity of this version of the algorithm is $\mathcal{O}(\#V \cdot \#E^2)$.

3 Bipartite graphs

Let us recall that a graph $G = (V, E)$ is *bipartite* if there is a decomposition of the set of vertices V into two subsets V_1, V_2, with $V_1 \cap V_2 = \emptyset$ and $V_1 \cup V_2 = V$, such that

$$\binom{V_1}{2} \cap E = \binom{V_2}{2} \cap E = \emptyset.$$

That is, there is no edge within V_1 and no edge within V_2, every edge of G has one endpoint in V_1 and the other in V_2.

Bipartite graphs are quite common, e.g., in applications where one wants to inspect relations between two groups of different objects. For example, employees and tasks, patients and drugs, processors and jobs, etc.

Although the definition of a bipartite graph is quite easy to understand, it would be quite unsuitable and difficult to use the definition to decide whether a graph is bipartite.

The following theorem gives us a simple characterisation of bipartite graphs, which is, moreover, easy to check: it can be done in linear time with respect to the number of vertices in the graph.

Theorem 1 *A graph G is bipartite if and only if there is no cycle of odd length in G.*

Example 3 The complete graph K_3 is not bipartite, since it has a cycle of length 3.

On the other hand, every tree is bipartite since it has no cycle, and thus in particular no odd cycles.



Figure 4: The complete graph K_3 (left) and a tree (right).

Let G be a bipartite graph. Let A_G be the adjacency matrix of G. One can easily see that there is a simultaneous permutation of rows and columns of A_G which transforms A_G into the following form:

$$A_G = \begin{pmatrix} O & B \\ B^T & O \end{pmatrix}.$$

Indeed, this simultaneous permutation of rows and columns corresponds to the permutation of vertices.

Example 4 Let $G = (V, E)$ be the graph on the left of Figure 5. The graph is bipartite with $V = V_1 \cup V_2$, where $V_1 = \{1, 4, 5\}$ and $V_2 = \{2, 3\}$. Its adjacency matrix is

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix}.$$

If we permute the names of vertices $2 \leftrightarrow 4$ and $3 \leftrightarrow 5$, which corresponds to a simultaneous switch of the 2nd and 4th row and column and 3rd and 5th row and column, we get the adjacency matrix

$$\left(\begin{array}{ccc|cc} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ \hline 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{array}\right) = \begin{pmatrix} O_3 & B \\ B^T & O_2 \end{pmatrix} \qquad \text{with} \qquad B = \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix},$$

corresponding to the graph on the right of Figure 5.

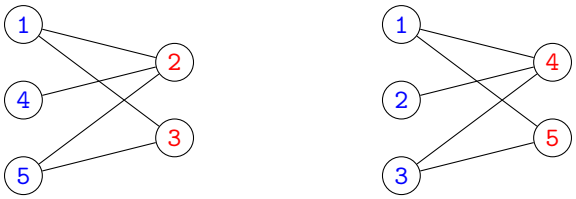


Figure 5: Permutation of vertices in a bipartite graph.

The zero square submatrices on the diagonal of A_G after the permutation correspond to the fact that there are no edges within V_1 and V_2.

4 Matching

Let $G = (V, E)$ be a graph. A *matching* in G is a set $M \subset E$ such that for every $e, f \in M$ with $e \neq f$ we have $e \cap f = \emptyset$. In other words, a matching is a set of edges without common vertices.

A vertex v is called *matched* by M (or M-*saturated*) if it is an endpoint of an edge of M. Otherwise, v is *unmatched* (or M-*unsaturated*).

A matching M is called *perfect* if every vertex in V is matched by M.

Example 5 There are a applicants to j different jobs. Each applicant has a subset of jobs in which they are interested. Each job opening can accept only one applicant, and each applicant can be assigned at most one job.

Finding an assignment of jobs to applicants in such a way that as many applicants as possible get a job corresponds to finding a maximum match in a bipartite graph $G = (A \cup J, E)$, where

- A is the set of applicants,
- J is the set of jobs,

and for each $a \in A$, $j \in J$ we have $\{a, j\} \in E$ if the applicant a is interested in the job j (see Figure 6).

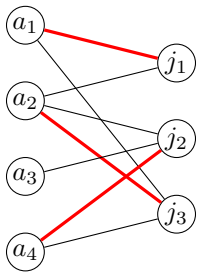


Figure 6: A bipartite graph with vertices representing applicants (left) and jobs (right) with a possible match (in red).

Example 6 We have a computer with 2 processors and a set of q jobs (computational). Some of these jobs can run simultaneously (for simplicity, let us assume that each job takes the same amount of time). To find the order of computation of jobs such that the total running time is minimised, we can proceed as follows.

Let us construct a graph with

- set of vertices = jobs,
- edges of the form $\{u, v\} \in E$ if jobs u and v can run simultaneously.

Then, a maximum matching in G gives the largest possible number of jobs that can be run simultaneously. If there are q jobs and the maximum matching has size $|M|$, the schedule requires $q - |M|$ time slots. Therefore, a maximum matching minimises the total running time.

A matching that cannot be extended to a larger matching is called *maximal*. A matching M in G is called *maximum* if for each matching M' in G we have $\#M \geq \#M'$. Note that a maximum matching is maximal, but not every maximal matching is a maximum.

Example 7 Let G be the graph represented in Figure 7. A maximal matching is represented on the left of the figure. No edge can be added to the matching to obtain a larger one.

A maximum matching is shown on the right of the same figure. Since all vertices are covered by it, the matching is perfect.



Figure 7: A maximal matching (left) and a maximum and perfect matching (right) for a graph.

5 Perfect matching

Obviously, not every graph contains perfect matchings. There are simple necessary conditions.

Proposition 1 *If G has a perfect matching, then $\#V$ is even.*

Proposition 2 *If $G = (V_1 \cup V_2, E)$ is a bipartite graph with a perfect matching, then $\#V_1 = \#V_2$.*

There are also some sufficient conditions, but most of them lead to high computational time of the decision procedure.

Recall that the neighbourhood of a vertex v in a graph $G = (V, E)$ is the set $\mathrm{N}_G(v) = \{u \in V \mid \{u, v\} \in E\}$. The *neighbourhood* of a set $S \subset V$ is defined as $\mathrm{N}_G(S) = \bigcup_{v \in S} \mathrm{N}_G(v)$.

Theorem 2 (Hall (1935)) *Let $G = (V_1 \cup V_2, E)$ be a bipartite graph. There is a matching M in G such that every $v \in V_1$ is matched by M if and only if for every $S \subset V_1$ we have $\#S \leq \#\mathrm{N}_G(S)$.*

Hall's Theorem 2 is also known as the *Marriage Theorem*. Imagine a dating app for heterosexual people used by a certain number of girls and a certain number of boys. If every group of she-users collectively likes at least as many he-users as there are girls in the group, then each girl can marry a boy she likes.

The condition in the theorem works for all bipartite graphs but it is not fast to check. Since it requires the inequality to be checked for all subsets of V_1, the time complexity is exponential in $\#V_1$.

Recall that a graph G is called regular if there exists a non-negative integer $k \in \mathbb{N}_0$ such that $\delta(G) = \Delta(G) = k$, i.e., every vertex is incident to exactly k edges.

Corollary 1 *Let G be a k-regular bipartite graph, with $k \geq 1$. Then G contains a perfect matching.*

The condition $\delta(G) = \Delta(G)$ in Corollary 1 is quite easy to check, but it is only valid for a special (quite small) class of graphs.

6 Augmenting paths

Let $G = (V, E)$ be a graph, and M a matching in G. A path $P = (v_i)_{i=0}^{k}$ in G is called M-*alternating* if for every $i \in \{1, \dots k-1\}$ we have

$$\{v_{i-1}, v_i\} \in M \quad \Leftrightarrow \quad \{v_i, v_{i+1}\} \notin M,$$

i.e., it is a path in which edges alternate between those in M and those not in M. An M-alternating path is M-*augmenting* if it both its endpoints are unmatched by M.

Example 8 Let G be the graph in Figure 8 and $M = \{\{\mathtt{a},\mathtt{b}\},\{\mathtt{d},\mathtt{e}\},\{\mathtt{g},\mathtt{h}\}\}$ a matching in G. The path $P_1 = (\mathtt{a},\mathtt{b},\mathtt{e},\mathtt{d},\mathtt{g},\mathtt{h})$ is M-alternating. The path $P_2 = (\mathtt{c},\mathtt{b},\mathtt{a},\mathtt{d},\mathtt{e},\mathtt{f})$ is both M-alternating and M-augmenting.

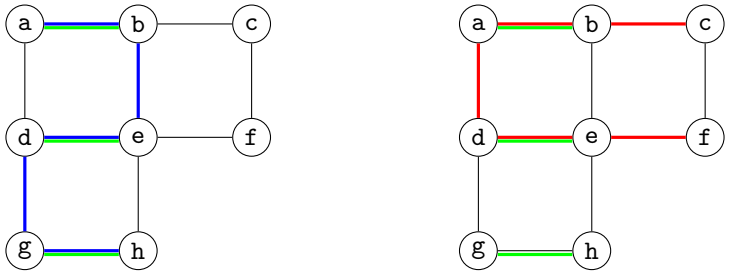


Figure 8: An M-alternating path (left) and an M-augmenting path (right).

An M-augmenting path $P = (V(P), E(P))$ in a graph G can be used to find a larger matching in G. Indeed, we can consider the matching $M' = M \triangle E(P) = (M \setminus E(P)) \cup (E(P) \setminus M)$. For simplicity, we will write $M \triangle P$ instead of $M \triangle E(P)$. That is, M' is constructed from M in the following way:

- edges both in $E(P)$ and M are discarded from M;
- edges in $E(P)$ but not in M are added to M;
- edges in M but not in $E(P)$ are left in M.

Obviously $\#M' = \#M + 1$. This implies that the absence of a M-augmenting path in G is a necessary condition for M to be maximum.

Example 9 Let G, M and P_2 as in Example 8. The match $M' = M \triangle P_2 = \{\{\mathtt{a},\mathtt{d}\},\{\mathtt{b},\mathtt{c}\},\{\mathtt{e},\mathtt{f}\},\{\mathtt{g},\mathtt{h}\}\}$ is also a matching in the graph (see Figure 9)

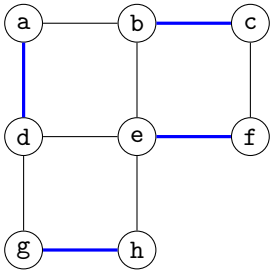


Figure 9: A matching in a graph.

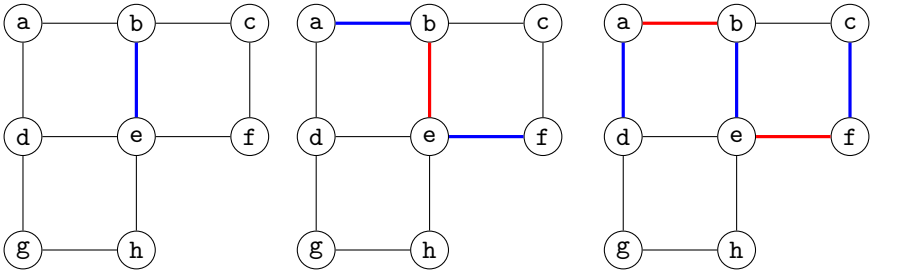
The following theorem states that this condition is also sufficient.

Theorem 3 (Berge (1957)) *Let $G = (V, E)$ be a graph, and M be a matching in G. Then, M is a maximum matching if and only if there is no M-augmenting path in G.*

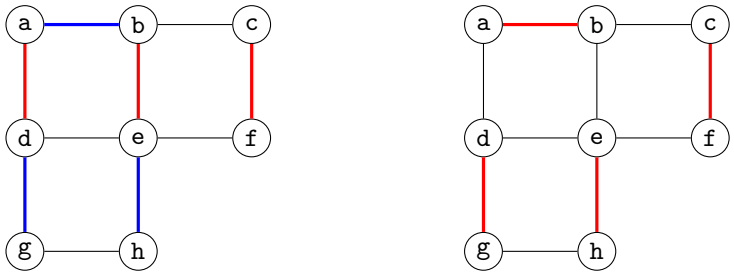
Thus, given a graph, we have a way to construct a maximum matching: we start with some matching M, e.g., the empty matching, and then we search for a M-augmenting path P. If such a path exists, we replace M by $M \triangle P$. We repeat the process until no augmenting path can be found.

Example 10 Let G again be the graph in Example 8. A possible sequence of matchings and corresponding augmenting paths, starting from the empty matching, is shown in Figure 10.

- We start with the empty matching $M_0 = \emptyset$. An M_0-augmenting path is given by the path $P_1 = (\mathtt{b}, \mathtt{e})$ (see Figure 10a).
- A larger matching is given by $M_1 = M_0 \triangle P_1 = \{\{\mathtt{b}, \mathtt{e}\}\}$. An M_1-augmenting path is given by $P_2 = (\mathtt{a}, \mathtt{b}, \mathtt{e}, \mathtt{f})$ (see Figure 10b).
- A larger matching is obtained as $M_2 = M_1 \triangle P_2 = \{\{\mathtt{a}, \mathtt{b}\}, \{\mathtt{e}, \mathtt{f}\}\}$. An M_2-augmenting path is given by $P_3 = (\mathtt{d}, \mathtt{a}, \mathtt{b}, \mathtt{e}, \mathtt{f}, \mathtt{c})$ (see Figure 10c).
- A larger matching is given by $M_3 = M_2 \triangle P_3 = \{\{\mathtt{a}, \mathtt{d}\}, \{\mathtt{b}, \mathtt{e}\}, \{\mathtt{c}, \mathtt{f}\}\}$. An M_3-augmenting path is given by $P_4 = (\mathtt{g}, \mathtt{d}, \mathtt{a}, \mathtt{b}, \mathtt{e}, \mathtt{h})$ (see Figure 10d).
- The matching $M_4 = M_3 \triangle P_4 = \{\{\mathtt{a}, \mathtt{b}\}, \{\mathtt{c}, \mathtt{f}\}, \{\mathtt{d}, \mathtt{g}\}, \{\mathtt{e}, \mathtt{h}\}\}$ is maximal. One can check that it is also a maximum matching and a perfect one.



(a) A path $(\mathtt{b}, \mathtt{e})$. (b) A path $(\mathtt{a}, \mathtt{b}, \mathtt{e}, \mathtt{f})$. (c) A path $(\mathtt{d}, \mathtt{a}, \mathtt{b}, \mathtt{e}, \mathtt{f}, \mathtt{c})$.



(d) A path $(\mathtt{g}, \mathtt{d}, \mathtt{a}, \mathtt{b}, \mathtt{e}, \mathtt{h})$. (e) A maximal matching.

Figure 10: A sequence of augmenting paths in a graph.

7 Hungarian method

Let us describe a systematic procedure for finding an augmenting path relative to a matching M in a given graph G. We first consider bipartite graphs and later extend the procedure to general graphs.

Let us start with a matching M in G. An *M-alternating tree* is a rooted tree whose root is unmatched by M and whose root-to-vertex paths are M-alternating (see Example 12).

If a root-to-vertex path ends with an unmatched vertex, distinct from the root, then it is an M-augmenting path.

Example 11 Let G and $M = \{\{\mathtt{a}, \mathtt{b}\}, \{\mathtt{d}, \mathtt{e}\}, \{\mathtt{g}, \mathtt{h}\}\}$ be the graph and the matching on the left of Figure 11 (see also previous lecture).

The tree $T(\mathtt{c})$ shown on the right of Figure 11 is an M-alternating tree. The path $(\mathtt{c}, \mathtt{b}, \mathtt{a}, \mathtt{d}, \mathtt{e}, \mathtt{f})$ is an M-augmenting path contained in the tree .

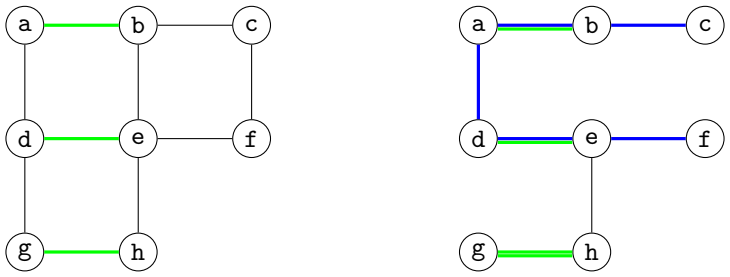


Figure 11: A graph G with a matching M (left) and an M-alternating tree and an M-augmenting path (right).

The following algorithm was developed by the Hungarian mathematician Egerváry (1931) and it is also known as the *Hungarian Algorithm* (such an algorithm can also be generalised to weighted graphs).

Hungarian Algorithm. Let us divide the vertices in the tree according to the parity of their level: even levels of an alternating tree are labeled S (*sudý*), while vertices of the odd levels are labeled L (*lichý*). We keep unlabeled the vertices that are currently in no alternating tree.

Initially, all vertices are unlabeled, and all edges are unexplored.

At each step, we have two options.

(S1) Choose an unlabeled vertex r that is unmatched by M, label it S, and make it the root of a new tree.

(S2) Choose an unexplored edge $\{u, v\}$, where u is labeled S, and explore it as follows (5 options, based on v):

a) if v is unlabeled and unmatched, we label v as L, and we have found an M-augmenting path consisting of the path from the root of u's tree to u followed by the edge $\{u, v\}$;

b) if v is unlabeled and matched, let w be the vertex matched with v by M, i.e., the vertex such that $\{v, w\} \in M$; we label v as L and w as S, and we add the edges $\{u, v\}$ and $\{v, w\}$ to the tree;

c) if v is labeled L, we do nothing;

d) if v is labeled S and belongs to a different tree from u, then an M-augmenting path is obtained by following the path from the root of u's tree to u, traversing the edge $\{u, v\}$, and then following the path from v to the root of v's tree;

e) if v is labeled S and belongs to the same tree as u, then the edge $\{u, v\}$ together with the appropriate portions of the two root-to-vertex paths, determines an odd M-alternating cycle, called a *blossom*.

There is a special procedure how for treating blossoms, but the basic version of the Hungarian algorithm is used only on bipartite graphs in which the blossom (odd cycle) cannot occur.

The search stops when an M-augmenting path is found, i.e., at Step 2a) or Step 2d) (and such a path can be used to construct a larger matching), or when there is no unexplored edge incident with an S-labeled vertex and no unmatched vertex remains available as the root of a new alternating tree.

Proposition 3 *Let G be a bipartite graph. If there is an M-augmenting path in G, the Hungarian algorithm finds it in $\mathcal{O}(m)$ time, where m is the size of G.*

The previous proposition immediately gives an $\mathcal{O}(m \cdot n)$ time algorithm for finding a maximum matching in a bipartite graph of order n and size m.

Example 12 Let G be the bipartite graph in Figure 12, with set of vertices $V = V_1 \cup V_2$, where $V_1 = \{a_1, a_2, a_3, a_4, a_5\}$ and $V_2 = \{b_1, b_2, b_3, b_4\}$. Consider the matching $M = \{\{a_2, b_1\}, \{a_3, b_3\}, \{a_4, b_2\}\}$ in G (left of Figure 12).



Figure 12: Two matchings in a bipartite graph.

The alternating forest can be constructed using the Hungarian algorithm, e.g., as follows (see also Figures 13a to 13g).

Step	Type	Vertices/Edges	Matching/Path	Figure
	initial		M	12 (left)
1^{st}	S1	$r = a_5$		13a
2^{nd}	S2b)	$u = a_5,\ v = b_3,\ w = a_3$		13b
3^{rd}	S1	$r = a_1$		13c
4^{th}	S2b)	$u = a_1,\ v = b_1,\ w = a_2$		13d
5^{th}	S2c)	$u = a_2,\ v = b_3$		13e
6^{th}	S2b)	$u = a_1,\ v = b_2,\ w = a_4$		13f
7^{th}	S2a)	$u = a_4,\ v = b_4$	$P = (a_1, b_2, a_4, b_4)$	13g
	augment		$M' = M \triangle E(P)$	12 (right)

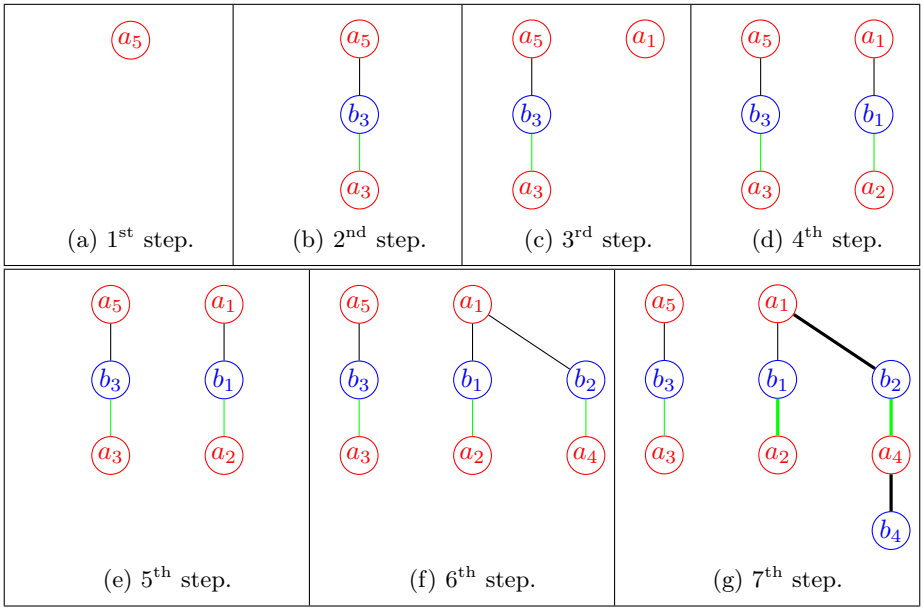


Figure 13: Steps of the Hungarian algorithm.

In the 7th step we have discovered an M-augmenting path $P = (a_1, b_2, a_4, b_4)$. We can obtain a larger matching by considering

$$M' = M \triangle E(P) = \{\{a_1, b_2\}, \{a_2, b_1\}, \{a_3, b_3\}, \{a_4, b_4\}\}$$

(see right of Figure 12).

Note that, since every vertex of V_2 is matched by M', we have $\#M' = \#V_2 = 4$. Therefore, M' is a maximum matching. In particular, M' is also maximal.

Let us extend the Hungarian augmenting-path algorithm in order to find a maximum matching in a graph that is not necessarily bipartite. In particular, we consider the case when a blossom is found. This new algorithm, used to find a maximum matching in a general graph, is called *Edmonds' Algorithm*.

Edmonds' Algorithm. Similarly to the Hungarian algorithm, we explore the graph and construct an M-alternating forest.

There is only one difference: since we consider a general graph, we can have steps of the type S2e). This corresponds to the occurrence of an odd M-alternating cycle in G, i.e., a blossom. When we detect such a blossom, we contract it into a single pseudovertex, label it S, and then we continue with the traversal. After an augmenting path is found in the contracted graph, we expand the blossom and lift the path to the original graph.

Example 13 Consider the graph $G = (\{\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}\}, \{\{\mathtt{A}, \mathtt{B}\}, \{\mathtt{A}, \mathtt{C}\}, \{\mathtt{B}, \mathtt{C}\}, \{\mathtt{B}, \mathtt{D}\}\})$ with the matching $M = \{\{\mathtt{A}, \mathtt{B}\}\}$ (see Figure 14).

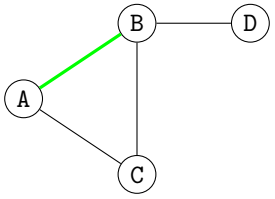


Figure 14: A graph G with an odd cycle.

The steps of Edmonds' Algorithm, starting from the matching M, are as follows.

Step	Type	Vertices/Edges	Matching/Path/Blossom	Figure
	initial		$M = \{\{\mathtt{A},\mathtt{B}\}\}$	14
1^{st}	S1	$r = \mathtt{C}$		15a
2^{nd}	S2b)	$u = \mathtt{C},\ v = \mathtt{A},\ w = \mathtt{B}$		15b
3^{rd}	S2e)	$u = \mathtt{C},\ v = \mathtt{B}$	$\{\mathtt{A},\mathtt{B},\mathtt{C}\} \to K$	15c
4^{th}	S2a)	$u = K,\ v = \mathtt{D}$	$P' = (K,\mathtt{D})$	15d
	lift		$P = (\mathtt{C},\mathtt{A},\mathtt{B},\mathtt{D})$	15e
	augment		$M' = \{\{\mathtt{A},\mathtt{C}\},\{\mathtt{B},\mathtt{D}\}\}$	15f

Table 1: A simple execution of the blossom algorithm.

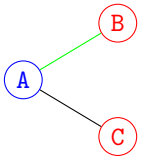
The lifted path $P = (\mathtt{C},\mathtt{A},\mathtt{B},\mathtt{D})$ is M-augmenting. Since the augmented matching

$$M' = M \triangle (P) = \{\{\mathtt{A},\mathtt{C}\},\{\mathtt{B},\mathtt{D}\}\}$$

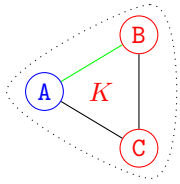
is perfect, it is also a maximum matching.



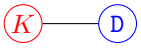
(a) 1^{st} step.



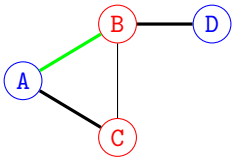
(b) 2^{nd} step.



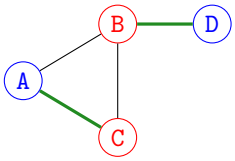
(c) 3^{rd} step.



(d) 4^{th} step.



(e) Lift of the blossom.



(f) A new matching.

Figure 15: Steps of Edmonds' Algorithm.

Exercise [Hungarian algorithm]. Using the Hungarian algorithm and starting from the empty matching $M_0 = \emptyset$, find a maximum matching M for the graph G in Figure 16

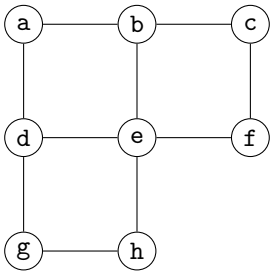


Figure 16: A graph with empty matching $M_0 = \emptyset$.

A perfect matching is also called a 1-*factor*, since it corresponds to a spanning 1-regular subgraph. When the set of edges of a graph can be partitioned into k different 1-factors, we say that the graph has a 1-factorisation.

Theorem 4 *Let $n \geq 1$. The edges of K_{2n} can be partitioned into $2n-1$ pairwise disjoint 1-factors.*

Example 14 The edges of the graph K_4 can be partitioned into three perfect matchings $M_1 = \{\{\mathtt{a},\mathtt{b}\},\{\mathtt{c},\mathtt{d}\}\}$ (left of Figure 17), $M_2 = \{\{\mathtt{a},\mathtt{c}\},\{\mathtt{b},\mathtt{d}\}\}$ (centre of Figure 17), and $M_3 = \{\{\mathtt{a},\mathtt{d}\},\{\mathtt{b},\mathtt{c}\}\}$ (right of Figure 17).

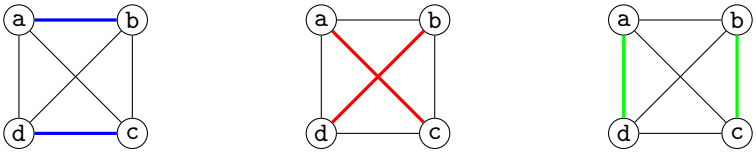


Figure 17: A 1-factorisation of K_4.