Graph Algorithms

François Pirot

December 9, 2021

Contents

About the course

1 Prerequisites

- We will recall the important notions about the structure of a graph in the first session; it will be important to be familiar with them for the next sessions.
- You should know what an algorithm is, how to prove that it is correct, and how to compute its complexity, in time and in space. You should be able to use classic algorithmic strategies such as greedy algorithms, divide and conquer, dynamic programming.
  Cf. Cormen *Introduction to Algorithms* if needed:
  `https://edutechlearners.com/download/Introduction_to_algorithms-3rd%20Edition.pdf`
- You should able to articulate and formalise a reasoning. There will be an important focus on the quality of formulation during the course.
- Basic knowledge of probabilities.

2 Goal

By the end of this course, you should be able to prove structural properties on graphs, and use them in order to solve advanced problems on graphs. To that end, you will have to understand and be able to apply a subset of methods that are classical in Graph Theory.

3 Evaluation

Contrôle continu : DM à mi-parcours. Evaluation finale : DS.

4 Rules

- Everyone is expected to participate actively during the exercise sessions. In particular everyone will go on the board, either from their initiative, or else will be designated at some point. This is the best moment to make mistakes. There is no shame in saying nonsense; on the contrary this is the best way to get rid of it.

  Tout le monde passe au tableau, soit volontairement, soit en étant désigné. Il n'est pas nécessaire d'avoir résolu l'exercice pour cela. C'est le meilleur endroit pour se tromper, il n'est pas grave d'y dire des bêtises ; au contraire c'est le meilleur moyen de les exorciser.

- During the exercise sessions, you can either work alone, or exchange with your direct neighbours. You should remain discrete while doing so in order to maintain a quality work place. During the correction, all are welcome to share their ideas or concerns.

  Pendant les phases d'exercice, on peut réfléchir seul ou à plusieurs avec ses voisins directs, le tout en restant le plus discret possible. Pendant la correction, tout le monde est encouragé à participer pour proposer des idées.

- DM: You are allowed to brainstorm with others and share your thoughts. However, every rendition must be personal; no plagiarism will be tolerated between each other's formulation.

  DM: Vous êtes autorisés à discuter entre vous et partager vos réflexions. Cependant, chaque rendu doit être personnel : pas de plagiat entre les rédactions de chacun.

Chapter 1

Introduction

1 Reminders of Graph Theory

1.1 Definitions

A *graph* G consists of a pair (V, E), where V is a finite set of *vertices*, and $E \subseteq \binom{V}{2}$ is the set of *edges*. When V and E are not properly introduced, we refer to them as $V(G)$ and $E(G)$. A graph is the mathematical object that represents the possible pairwise interactions within a finite set of objects. The course focuses on *loopless*, *simple* graphs, meaning that every edge involves two different vertices, and that there is no more than one edge between two given vertices.

We need several definitions to analyse the structure of a graph: neighbour, neighbourhood ($N(v)$, $N[v]$, $N(X)$, $N[X]$), incidence, degree (minimum δ, maximum Δ, average ad), regular graphs.

Theorem 1 (Handshaking Lemma)**.** *For every graph $G = (V, E)$,*

$$\sum_{v \in V} \deg(v) = 2|E|.$$

1.2 Subgraph, induced subgraph

1.2.1 Definition

Let $G = (V, E)$ be a graph.

- $H = (V', E')$ is a *subgraph* of G if $E' \subseteq E$ and $V' \subseteq V$.
- For every $X \subseteq V$, the subgraph $G[X] = (V', E')$ of G induced by X is such that $V' = X$ and E' is the set of edges $e \in E$ such that $e \subseteq X$.
- H is an *induced subgraph* of G if there exists $X \subseteq V$ such that $H = G[X]$.

1.2.2 Important subgraphs

Clique, independent set, cycle, (induced) path. Notations K_n, C_n, P_n.
Define $\omega(G)$, $\alpha(G)$, girth, distance, eccentricity, radius, diameter.
Note: A shortest path is always induced.
Complexity of computing α, ω, a longest path/cycle.

1.2.3 Operations

Let $G = (V, E)$ be a graph.

- vertex deletion: $G - v = G \setminus v = G[V \setminus \{v\}]$.
- edge deletion $G - e = G \setminus e$, edge addition $G + e$.
- edge contraction G/e.

1.2.4 Heredity

A graph property $\pi(\cdot)$ is *hereditary* if $\pi(G) \implies \pi(H)$ for every induced subgraph H of a given graph G.

Examples Let k be a fixed integer.

- Having maximum degree at most k is hereditary.
- Having clique number at most k is hereditary.
- Having independence number at most k is hereditary.
- Having girth at least k is hereditary.
- Having bounded diameter is NOT hereditary (the diameter may increase or decrease in an induced subgraph).
- Having bounded minimum degree is NOT hereditary (the minimum degree may increase or decrease in an induced subgraph).
- Having bounded average degree is NOT hereditary (the average degree may increase or decrease in an induced subgraph).

We define the hereditary versions of the minimum and average degrees. Given a graph G, the *degeneracy* $\delta^*(G)$ of G is the maximum over all (induced) subgraphs H of G of $\delta(H)$, and the *maximum average degree* $\mathrm{mad}(G)$ of G is the maximum over all (induced) subgraphs H of G of $\mathrm{ad}(H)$.

Theorem 2 (Cf. Exercise 2.1 of TD1)**.** *For every graph G,*

$$\delta^*(G) \leq \mathrm{mad}(G) \leq 2\delta^*(G).$$

1.3 Connectivity

1. Given two vertices $u, v \in V(G)$, let us denote $u \sim_G^* v$ whenever there exists a path in G between u and v. It is routine to show that $\sim_G^*$ is an equivalence relation on $V(G)$, which is the transitive closure of the adjacency relation $\sim_G$. The equivalence classes induced by $\sim_G^*$ are called the *connected components* of G. In other words, a connected component of G is a maximal subset of vertices that can be pairwise joined by some path. If G contains only one connected component, we say that G is *connected*, otherwise we say that G is *disconnected*.
2. When G is connected, a *cut* in G is a subset of edges $X \subseteq E(G)$, the removal of which creates two or more connected components. Alternatively, a cut may be described by a bipartition of the vertices of G; it then consists of all the edges lying between the two parts of the partition. A *vertex-cut* in G is a subset of vertices $X \subseteq V(G)$ such that the subgraph induced by $V(G) \setminus X$ is disconnected. A cut of size 1 is called a *bridge*, and a vertex-cut of size 1 is called a *cut-vertex*.
3. For any $k \geq 1$, we say that G is *k-edge-connected* if all cuts of G have size at least k. We say that G is *k-connected* if G contains at least $k + 1$ vertices, and all vertex-cuts of G have size at least k.

1.4 Trees

1.4.1 Properties

A *forest* is an acyclic graph. A connected forest is called a *tree*.

Theorem 3. *Let $G = (V, E)$ be a graph. The following statements are equivalent.*

(i) *G is a tree.*

(ii) *G is connected, and every edge of G is a bridge.*

(iii) *G is connected and $|E| = |V| - 1$.*

(iv) *G is acyclic and $|E| = |V| - 1$.*

Proof. We prove the equivalence with a cycle of implications.

- (i) $\implies$ (ii): Assume for the sake of contradiction that $G \setminus e$ is connected for some edge $e = uv \in E$. Let P be a shortest path from u to v in $G \setminus e$, then $P + e$ is a cycle in G, a contradiction.
- (ii) $\implies$ (iii): By induction on the number m of edges. If $m = 0$, then G is a single vertex. Otherwise, let $e \in E$; $G \setminus e$ has two connected components on which we apply induction.
- (iii) $\implies$ (iv): By induction on the number n of vertices. If $n = 1$, then G is obviously acyclic. Assume now that $n \geq 2$. By the pigeonhole principle, there exists a vertex v of degree 1. By the induction hypothesis, $G \setminus v$ is acyclic, and so is G since v has degree 1 and therefore is contained in no cycle.
- (iv) $\implies$ (i): Assume for the sake of contradiction that G is not connected. Let $G_i = (V_i, E_i)$ be its connected components; by the pigeonhole principle there exists i such that $|E_i| \geq |V_i|$. Since (i) $\implies$ (iii), we infer that G_i is not a tree, and as it is connected it must contain a cycle. This contradicts the fact that G is acyclic.

□

A forest $F = (V, E)$ with k connected components satisfies $|E| = |V| - k$, so its average degree $\frac{2|E|}{|V|}$ is less than 2. Said otherwise, a graph with average degree at least 2 must contain a cycle. Another consequence is that the class of 1-degenerate graphs (graphs G such that $\delta^*(G) \leq 1$) consists exactly of the forests.

1.4.2 Spanning trees

Given a connected graph $G = (V, E)$, a *spanning tree* of G is a subgraph of G on the vertex set V, which is a tree.

Theorem 4. *A graph G is connected if and only if it contains a spanning tree.*

Proof. If G is not a tree, let $e = uv$ be an edge of a cycle C of G. Then e is not a bridge in G, since otherwise u and v would belong to different components in $G \setminus e$, yet there is a path that joins them, namely $C \setminus e$, a contradiction. So $G \setminus e$ is connected. Repeating that operation ultimately yields a tree, which is a spanning tree of G.

Conversely, if G contains a spanning tree T, then all pairs of vertices are connected through their path in T.

□

1.4.3 Rooted trees

A *rooted tree* T is a tree given with a special vertex $r \in V(T)$, the *root* of T. In a rooted tree T, the vertex set $V(T)$ is usually considered together with a partition $(V_0, \dots, V_p)$, where $V_0 = \{r\}$, and V_i is the set of vertices at distance i from r in T, for any $i \geq 1$. Each V_i is called the *layer* at depth i of T. The *depth* p of the tree is the maximum depth of its layers, which corresponds to the eccentricity of its root r.

Given a vertex $v \in V_i(T)$ at depth i in a rooted tree T, the *children* of v are its neighbours at depth $i + 1$, that is $N(v) \cap V_{i+1}$. When $i > 0$, the vertex v has exactly one other neighbour, which lies at depth $i - 1$; we call it the *parent* of v.

1.4.4 Exploration trees

A *Breadth First Search tree* (or BFS-tree) of a given graph G is a spanning tree T rooted in some vertex $v \in V(G)$, such that the layer at depth i in T consists of the set of vertices at distance i from v in G.

A *Depth First Search* is an algorithm that constructs a spanning tree T of G, beginning in a root vertex $v \in V(G)$, by iteratively adding a neighbour of the last vertex added in T that still contains some neighbours not in T, until T spans $V(G)$. A *Depth First Search tree* (or DFS-tree) of G is a spanning tree T that can be obtained with a Depth First Search in G.

Chapter 2

Graph colouring

1 Definition

Let $G = (V, E)$ be a graph, and k an integer. A *proper k-colouring* of G can be defined as

- a partition of V into k independent sets;
- a mapping $c\colon V \to [k]$ such that $c(u) \neq c(v)$ for every edge $uv \in E$.

If there exists a proper k-colouring of G, we say that G is *k-colourable*. The *chromatic number* $\chi(G)$ of G is the minimum integer k such that G is k-colourable; if $\chi(G) = k$ then we say that G is *k-chromatic*.

We say that G is *bipartite* if G is 2-colourable, i.e. $\chi(G) \leq 2$.

2 Noticeable properties

Let G be a graph.

- $\chi(G) \geq \omega(G)$.
- $\chi(G) \geq \frac{|V(G)|}{\alpha(G)}$.
- More generally, we define the *Hall ratio* $\rho(G)$ of G as $\max\limits_{H \subseteq G} \frac{|V(H)|}{\alpha(H)}$. Note that $\rho(G) \geq \omega(G)$.
- $\chi(\cdot)$ is hereditary, therefore $\chi(G) \geq \rho(G)$.
- For every fixed integer $k \geq 3$, the problem of deciding whether $\chi(G) \leq k$ given a graph G is NP-complete.

Conjecture 1 (Reed's Conjecture)**.** *For every graph G,*

$$\chi(G) \leq \frac{\omega(G) + \Delta(G) + 1}{2}.$$

3 Greedy colouring

Theorem 5. *The proper colouring returned by the greedy colouring applied on any given graph G uses at most $\Delta(G) + 1$ colours.*

Proof. The variable k counting the number of colours used by c during the execution of the algorithm is incremented only when $c(N(v)) = [k]$, for some vertex $v \in V(G)$. When this happens, in particular, $\Delta(G) \geq |N(v)| \geq k$. So, if k reaches the value $\Delta(G) + 1$, it cannot be further incremented. $\square$

Algorithm 1: Greedy colouring

```
Data: Some graph $G$ with $V(G) = \{v_1, \ldots, v_n\}$
Result: $c$ is a proper $k$-colouring of $G$
$k \leftarrow 1, i \leftarrow 1$
for $i$ from $1$ to $n$ do
    if $c(N_G(v_i)) = [k]$ then
        $k \leftarrow k + 1$
        $c(v_i) \leftarrow k + 1$
    else
        $c(v_i) \leftarrow \min \quad [k] \setminus c(N_G(v_i))$
    end
end
return $c$
```

Complexity The complexity of the greedy colouring algorithm is $O(m)$ where m is the number of edges in G. Indeed, at each step i, the algorithm performs $O(\deg(v_i))$ operations.

Importance of the order The order that we choose for the greedy algorithm matters. If $v_1, \ldots, v_n$ is an order obtained by repeatedly extracting vertices of minimum degree from G, then the greedy colouring applied with the reverse ordering uses at most $\delta^*(G) + 1$ colours.

4 Brooks' Theorem

The bound stated by Theorem 5 is tight, as it is reached by complete graphs and odd cycles. It turns out that these are the only graphs for which this bound is tight.

Theorem 6 (Brooks)**.** *Let G be a connected graph. If G is neither complete nor an odd cycle, then*

$$\chi(G) \leq \Delta(G).$$

Before proceeding with the proof of Theorem 6, we need to define the block decomposition of a graph.

Definition. Let G be a connected graph.

- A *block* in G is a maximal 2-connected subgraph of G.
- A *block decomposition* of G is the set of blocks $B_1, \ldots, B_r$ in G. These blocks are connected through articulation points in G, and form a tree-like structure: every cycle in G is entirely contained in one of its blocks.
- An *end-block* in G is a block that contains only one articulation point. It plays the role of a leaf in the block decomposition of G.

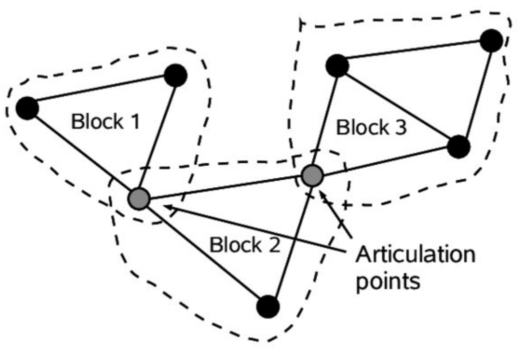

Figure 2.1: In this block decomposition, Block 1 and Block 3 are end-blocks.

We will use the following lemma.

Lemma 1. *Let G be a non-complete 2-connected graph of minimum degree at least 3. Then there exists an induced path $a - v - b$ such that $G - a - b$ is connected.*

Proof. By the assumptions on G, there exists a vertex x of degree at least 3, and less than $|V(G)| - 1$.

- If $G - x$ is 2-connected, then let b be a vertex at distance 2 from x in G, let v be a common neighbour of b and x, and let $a = x$. Then $a - v - b$ is an induced path in G, and since $G - a$ is 2-connected, $G - a - b$ is connected.
- If $G - x$ is not 2-connected, we consider a bloc decomposition of G, and let B_1, B_2 be 2 end-blocs of that decomposition, of articulation points z_1 and z_2, respectively. Since G is 2-connected, $B_1 - z_1$ and $B_2 - z_2$ are connected through x, so x has a neighbour $a \in B_1 - z_1$ and a neighbour $b \in B_2 - z_2$. Letting $v = x$ yields the solution.

□

We can now prove Theorem 6.

Proof of Theorem 6. If $\Delta(G) = 2$, then by the assumptions G is an even cycle, so G is 2-colourable. So we may assume now that $\Delta(G) \geq 3$.

If G is 2-connected, we apply Lemma 1 in order to find vertices v, a, b such that $a - v - b$ is an induced path in G, and $G - a - b$ is connected. Since a and b are not neighbours, we may colour both with colour 1. We let T be a spanning tree of $G - a - b$ rooted in v, and let $\prec$ be an order of the vertices of $G - a - b$ such that $x \prec y$ if y is the parent of x in T. We apply the greedy algorithm on $G - a - b$ with that order and obtain a proper colouring c of G. For every vertex $u \neq v$, there is one uncoloured neighbour of u when the algorithm assigns a colour to u, so $c(u) \leq \deg(u)$. When v receives a colour, it has two neighbours (a and b) with the same colour, so $c(v) \leq \deg(v)$. We conclude that c uses at most $\Delta(G)$ colours.

If G is not 2-connected, let x be an articulation point of G, and $G_1, \ldots, G_r$ the connected components of $G - x$. We may colour each graph $G_i + x$ inductively, and permute the colours so that all the colours agree on the colour of x. The union of those colourings yields a proper colouring of G with at most $\Delta(G)$ colours. □

Chapter 3

Complexity

1 Algorithmic aspects in the previous chapters

In this section, we formalise and optimise the complexity of the algorithms described in order to solve problems from the previous chapters.

1.1 Computing a degeneracy ordering

Given a d-degenerate graph G, a degeneracy ordering $\prec$ of G is obtained by repeatedly extracting vertices of minimum degree from G, so that the number of neighbours u of a vertex $v \in V(G)$ with $v \prec u$ is at most d.

Such an ordering can be found in time $O(m)$ (where $m \coloneqq |E(G)|$), by using a bucket-queue. We compute in time $O(m)$ a table D such that $D[i]$ contains the list of vertices in G of degree i, and we initialise a variable $d_v \coloneqq \deg_G(v)$ for every vertex $v \in V(G)$. We then repeat n times the following sequence of instructions.

- Find the smallest i such that $D[i]$ is non-empty;
- Extract the first element x of $D[i]$: this is the next element in the ordering;
- For every neighbour y of x not yet extracted, remove y from $D[d_y]$ and add it to $D[d_y - 1]$, then decrement d_y.

By using a suitable implementation of the bucket-queue, it is possible to make all those operations in time $O(m)$.

1.2 Computing a block decomposition

We want to find a block decomposition of a connected graph G on m edges. Let T be a DFS tree of G, rooted in an arbitrary vertex $r \in V(G)$. We note that a vertex $v \in V(G) \setminus \{r\}$ is an articulation point in G if and only there exists a child u of v in T such that the subtree of T rooted in u has no back-edge to an ancestor (strict) of v (each child u of v with that property is contained in a distinct block in G). On the other hand, the root vertex r is an articulation point in G if and only if it has degree at least 2 in T (every child of v is contained in a distinct block in G).

Using this property, we can find the block decomposition of G in time $O(m)$.

Algorithm 2: Articulation Points

```
Data: $G = (V, E)$: graph on $n$ vertices, $r \in V(G)$: root vertex
Result: DFS tree of $G$ with the articulation points marked
children ← [[] for i from 1 to n]
depth ← [Void for i from 1 to n]
low ← [Void for i from 1 to n]
Function Exploration($v, d$):
    depth[v] ← $d$
    low[v] ← $d$
    foreach $u \in N(v)$ do
        if depth[u]=Void then
            children[v].append(u)
            Exploration($u, d+1$)
        end
        low[v] ← min(low[u], low[v])
        if $v \neq r$ and low[u] $\geq$ depth[v] then
            Mark $v$ as an articulation point
        end
    end
end
Exploration($r, 0$)
if children[r].size() $> 1$ then
    Mark $r$ as an articulation point
end
```

1.3 Complexity of Brooks' colouring

Let G be a connected graph of maximum degree Δ. Let us compute a rooted block-decomposition T of G in time $O(|E(G)|)$.

- First assume that T contains at least 2 blocks. For every block B of G, the maximum degree of $G[B]$ is at most $\Delta - 1$, so one can colour $G[B]$ greedily in any order with Δ colours. We simply colour the blocks of G in a DFS ordering with respect to T, where the first coloured vertex in each block is the articulation point that is shared with the parent block.

- Assume now otherwise that G is 2-connected. In order to find an induced P_3 with extremities a, b such that $G \setminus \{a, b\}$ is connected, we need to find a vertex x that is not universal, and compute a block decomposition of $G \setminus x$. This can be done in time $O(|E(G)|)$. The rest of the operations can be done in time $O(|E(G)|)$.

Overall, one can compute a Δ-colouring of a connected graph G of maximum degree Δ that is neither complete nor an odd cycle in time $O(m)$, where $m = |E(G)|$.

2 P and NP

2.1 Definitions

A decision problem is a question whose answer is either "yes" or "no". Such a problem belongs to the class P if there is a polynomial-time algorithm that solves any instance of the problem in polynomial time. It belongs to the class NP if, given any instance of the problem whose answer is "yes", there is a certificate validating this fact which can be checked in polynomial time (in particular this certificate should have a polynomial size); such a certificate is said to be succinct. It is immediate from those definitions that P $\subseteq$ NP.

For instance, the problem of determining whether a graph contains a Hamiltonian cycle is in NP, since a Hamiltonian cycle is a succinct certificate if the answer is “yes”.

Conjecture 2.

$$P \neq NP$$

Conjecture 2 is widely(but not universally) believed to be true, which means that there are problems in NP which cannot be solved in polynomial time. If the conjecture is true, then HAMILTONIANCYCLE would be such a problem, among many others that we call *NP-complete problems*. Informally speaking, those are the hardest problems in the class NP.

2.2 Polynomial reduction

A common approach to problem-solving is to transform the given problem into one whose solution is already known, and then convert that solution into a solution of the original problem. Of course, this approach is feasible only if the transformation can be made rapidly. The concept of polynomial reduction captures this requirement. A polynomial reduction of a problem P to problem Q is a pair of polynomial-time algorithms which transforms each instance X of P to an instance Y of Q, and the other which transforms a solution for the instance Y to a solution for the instance X. If such a reduction exists, we say that P is polynomially reducible to Q, and write $P \preceq Q$. The significance of polynomial reducibility is that if $P \preceq Q$, and if there is a polynomial-time algorithm for solving Q, then this algorithm can be converted into a polynomial-time algorithm for solving P. This means that

$$P \preceq Q \text{ and } Q \in \mathrm{P} \implies P \in \mathrm{P} \tag{$\star$}$$

Observe that, since the solution of a decision problem is either “yes” or “no”, the second algorithm which transforms a solution for the instance Y to a solution for the instance X, is trivial: either it is identity (the solution to X is “yes” if and only if the answer to Y is “yes”) or the negation (the answer to X is “yes” if and only if the answer to Y is “no”).

2.3 NP-complete problems

We have just seen how polynomial reductions may be used to produce new polynomial-time algorithms from existing ones. By the same token, polynomial reductions may also be used to link “hard” problems, ones for which no polynomial-time algorithm exists, as can be seen by taking the contrapositive of ($\star$) :

$$P \preceq Q \text{ and } P \notin \mathrm{P} \implies Q \notin \mathrm{P}.$$

This observation led Cook and Levin to define a special class of seemingly intractable decision problems, the class of NP-complete problems. Those problems are maximal with respect to $\preceq$ in NP. So, given an NP-complete problem P, every problem $P' \in$ NP is polynomially reducible to P. We denote by NPC the class of NP-complete problems.

It is by no means obvious that NPC is non-empty. On the other hand, if we now that some problem P is NP-complete, then we deduce that every problem Q such that $P \preceq Q$ is NP-complete as well.

Theorem 7 (Cook)**.** *The problem SAT is NP-complete.*

Building from that result, a large list of problems have been shown to be NP-complete. Among those, 3-SAT, INDEPENDENTSET, HAMILTONIANPATH, COLOR, 3-COLOR, ...

2.4 Reduction from 3-SAT to 3-Color

Let $C_1, \ldots, C_r$ be the clauses of a SAT instance X. We are going to construct a graph $G(X)$ such that $G(X)$ is 3-colourable if and only if X is satisfiable. We begin with a triangle with vertices labelled True, False, and Neutral. For every literal x that appears in X, we create a vertex labelled x, and we link it to the vertices labelled Neutral and $\overline{x}$. So in a 3-colouring of G, every literal-vertex will be coloured with the colour of the True-vertex, or that of the False-vertex, and the colour of opposite literals are different.

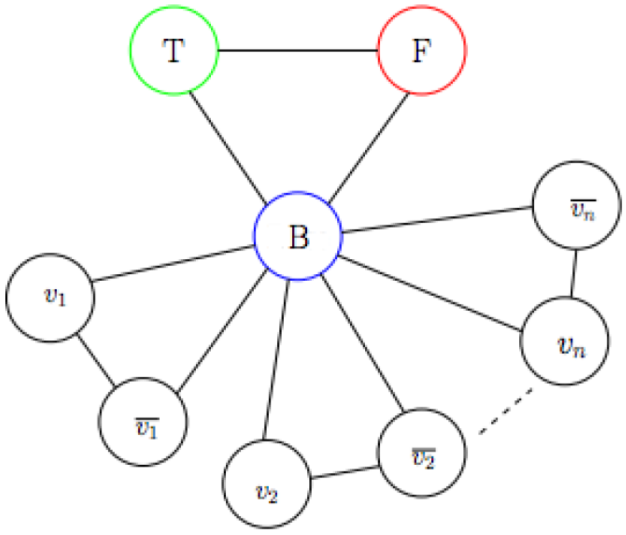

Figure 3.1: Pattern for all literals

Then, for each clause $a \vee b \vee c$, we construct the pattern depicted in Figure 3.2. If the vertices a and b both have the False colour, then the vertex $a \vee b$ must have the False colour as well. On the other hand, if a or b has the True colour, then $a \vee b$ may have the True colour as well (up to local recolouring). We force the $a \vee b \vee c$ vertex to have the True colour, which means that one of a, b, or c must have the True colour.

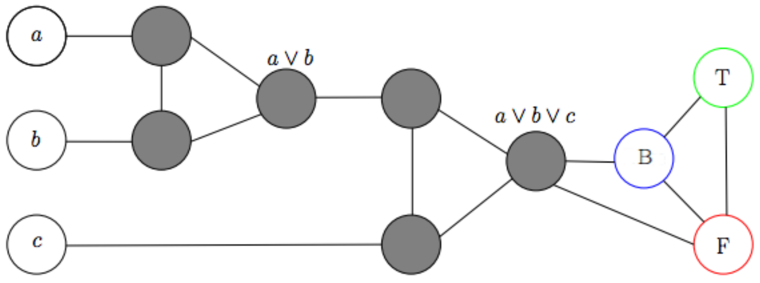

Figure 3.2: Pattern for each clause

If $G(X)$ is 3-colourable, then the colour given to each literal can be directly translated to a truth value, and that assignation satisfies the instance X. On the other hand, if X is satisfiable, then colouring each literal vertex with the colour of its truth value yields a partial 3-colouring of G that can be completed to a proper 3-colouring of G.

3 FPT algorithms

An algorithm P of parameter k is Fixed Parameter Tractable (FPT) if there exists a parameter $\pi()$ on its inputs such that, on an input X of size n with $\pi(X) \leq k$, the complexity of P is $f(k) \cdot n^{O(1)}$, for some function f. In general, f can be extremely large: there exist problems for which we need $f(k) > 2^{2^{\cdot^{\cdot^{\cdot^{k}}}}}$, for any tower of exponentials on the right-hand side.

Given an NP-complete problem P, we can seek for a parameter that makes it FPT. Finding such a parameter lets us describe large classes of instances for which the problem can be polynomially solved. It also helps us better understand where the complexity of the problem comes from.

For instance, the problem INDEPENDENTSET of deciding whether a graph G contains an independent set of size k is not known to be FPT in k; the best known complexity is $n^{O(k)}$. However, it is FPT for the parameter given by the couple $(\delta^*(G), k)$; one can solve it in time $O((\delta^*(G)+1)^{k+2}n)$.

Algorithm 3: IndependentSet

```
Data: $G$: $t$-degenerate graph, $k$: integer
if $k > |V(G)|$ then
    return False
end
if $k = 0$ or $|E(G)| = \varnothing$ then
    return True
end
$u_0 \leftarrow$ vertex of degree $d \leq t$
$u_1, \dots, u_d \leftarrow$ neighbours of $u_0$
for $i$ from $0$ to $d$ do
    $G_i \leftarrow G \setminus N[u_i]$
end
return $\bigvee_{i=0}^{d}$ IndependentSet($G_i$, k-1)
```

We prove that the algorithm `IndependentSet` is correct. The correctness is trivial if $k = 0$ or $k > |V(G)|$. Now, if G contains an independent set of size $k \geq 1$, then it contains an independent set of size k that contains one of the u_i's. Take a maximum independent set (so of size $\geq k$); by maximality it is a transversal of the sets $(N[v])_{v \in V(G)}$. Up to removing some vertices not in $N[u_0]$, this yields an independent set of G that contains a vertex in $N[u_0]$, and has size exactly k. Let I be an independent set of G of size k, then $I \setminus x$ is an independent set of $G \setminus N[x]$ of size $k-1$, for every $x \in I$. We conclude that G contains an independent set of size k if and only if $G \setminus u_i$ contains an independent set of size $k-1$, for some $0 \leq i \leq d$. This proves that the algorithm is correct.

The complexity of the algorithm, apart from the recursive calls, is dominated by the constructions of the graphs G_i, of cost $(d+1) \times O(n+m) = O(t^2 n)$. Let a_k be the maximum number of recursive calls made by a call to `IndependentSet(G,k)`, then $a_k \leq 1 + (t+1)a_{k-1}$. This implies that

$$a_k \leq 1 + (t+1) + (t+1)^2 + \ldots + (t+1)^k a_0 = \sum_{i=0}^{k} (t+1)^i.$$

If $t = 0$, then G is 0-degenerate, which means that $E(G) = \varnothing$, hence the algorithm stops in time $O(1)$. Otherwise, if $t > 0$, we obtain $a_k \leq \frac{(t+1)^{k+1}}{t}$. We conclude that the overall complexity is $O((t+1)^{k+1}tn) = O((t+1)^{k+2}n)$.

Chapter 4

Matchings

1 Presentation

1.1 Definitions

Given a graph G, a *matching* M of G is a set of non-incident edges (two different edges in M have distinct extremities). It is possible to have an equivalent definition with the help of *line graphs*. Given a graph $G = (V, E)$, the line graph of G, denoted $L(G)$, is the graph with vertex set E, and such that e_1e_2 is an edge in $L(G)$ for every pair of edges $e_1, e_2 \in E$ that are incident in G. Hence, a matching in G is an independent set in $L(G)$.

Given a subset of vertices $X \subseteq V(G)$, and a matching M of G, we say that M *saturates* X if $X \subseteq V(M)$. The vertices in a saturated set are *covered* by M. We say that M is a *perfect matching* of G it it saturates $V(G)$. Note that a perfect matching of G is of size $\frac{|V(G)|}{2}$.

More generally, we denote $\nu(G)$ the size of a maximum matching in G. In this chapter, we will see among other results that computing $\nu(G)$ can be done in polynomial time.

1.2 An example of application

There are many real world problems that can be formed as Bipartite Matching. For example, consider the following problem.

There are M job applicants and N jobs. Each applicant has a subset of jobs that he/she is interested in. Each job opening can only accept one applicant and a job applicant can be appointed for only one job. Find an assignment of jobs to applicants in such that as many applicants as possible get jobs.

1.3 Alternating paths

Let G be a graph, and M a matching of G. An *M-alternating path* of G is a path whose edges are alternatively in M and in $E(G) \setminus M$. If both extremities of an M-alternating path P are not covered by M, then P is an *M-augmenting path* of G.

If there exists an M-augmenting path P, then we can use P in order to obtain a larger matching M', by taking the symmetric difference $M' := M \bigtriangleup E(P)$. Therefore, a necessary condition for M to be a maximum matching in G is that there exists no M-augmenting paths in G. It turns out that this is also a sufficient condition.

Theorem 8 (Berge). *Let G be a graph, and M a matching of G. Then M is a maximum matching if and only if G contains no M-augmenting path.*

Proof. There remains only to prove that if M is not a maximum matching of G, then G contains an M-augmenting path. Let M' be a maximum matching of G, and let H be the subgraph given by the symmetric difference $M \bigtriangleup M'$. This is a subgraph of G of maximum degree 2, hence each connected component of H is either a path of a cycle, the edges of which alternate between M and M'. Therefore the cycles are all even. Since M' contains more edges than

M, there exists a connected component of H which has more edges in M' than in M. This is an M-augmenting path. □

Berge's Theorem states that every maximal matching is also maximum. Hence it is possible to construct a maximum matching of any graph with a greedy algorithm that begins with any matching M, and repeatedly looks for an M-augmenting path, and increases the size of M if it finds one, or returns M otherwise.

The main complexity of that algorithm comes from the cost of finding an M-augmenting path if one exists. There are various methods for that operation; for instance one based of flows in graphs has complexity $O(|E(G)|)$.

2 Matchings in bipartite graphs

When we work in bipartite graphs, we can exploit their structure in order to improve what we have seen in the previous Section.

2.1 Hall's condition

Given a bipartite graph $G = (U, V, E)$ where $|U| \leq |V|$, the maximum size of a matching M of G is at most $|U|$. We want to know whether there exists a matching M that saturates U. If there exists such a matching M, then for every subset $X \subseteq U$, the edges of M link X to as many vertices in V. Hence a necessary condition for the existence of a matching that saturates U is that $|N(X)| \geq |X|$, for every $X \subseteq U$. It turns out that this is also a sufficient condition.

Theorem 9 (Hall)**.** *Let $G = (U, V, E)$ be a bipartite graph. Then G contains a matching that saturates U if and only if $|N(X)| \geq |X|$ for every $X \subseteq U$.*

Proof. Let M be maximum matching in G. If M does not saturate U, let u_0 be an uncovered vertex. We consider the following algorithm.

```
Algorithm 4: Finding a set that contradicts Hall's condition
  Data: $M$: maximum matching of $G$, $u_0$: uncovered vertex
  $X \leftarrow \{u_0\}$
  $Y \leftarrow N(u_0)$
  while $|Y| \geq |X|$ do
      $X \leftarrow N_M(Y) \cup \{u_0\}$
      $Y \leftarrow N(X)$
  end
  return $X$
```

It is straightforward that the out condition of the While loop ensures that the algorithm returns a correct solution if it terminates.

Let us assume that there are at least $i \geq 1$ iterations of the While loop. Let (X_j, Y_j) be the value of (X, Y) after $j \leq i$ iterations of the While loop. No vertex $y \in Y_j$ is uncovered by M, otherwise there is an M-augmenting path from u_0 to y, so $|N_M(Y_j)| = |Y_j|$. We have $|Y_{i-1}| \geq |X_{i-1}|$ by assumption, and hence $|X_i| = |Y_{i-1}| + 1 \geq |X_{i-1}| + 1$. So, by induction, $|X_i| \geq i + 1$. We conclude that the algorithm stops after at most $|U| - 1$ iterations of the While loop, since X is always contained in U.

□

2.2 The Hungarian Method

By relying on Hall's Theorem, it is possible to find a matching that saturates U in a bipartite $G = (U, V, E)$ if one exists, or find a set violating Hall's condition otherwise. This algorithm works in polynomial time and is known as the *Hungarian Method*.

Algorithm 5: Hungarian Method

```
Data: $G = (U, V, E)$: bipartite graph
$M \leftarrow$ any matching of $G$
while $M$ does not saturate $U$ do
    $u_0 \leftarrow$ uncovered vertex
    $X \leftarrow \{u_0\}$
    $Y \leftarrow \varnothing$
    $T \leftarrow$ Tree($u_0$)
    repeat
        if $N(X) = Y$ then
            return $X$: violates Hall's condition
        end
        $v \leftarrow$ vertex from $N(X) \setminus Y$
        $u \leftarrow$ vertex from $N(v) \cap X$
        if $v$ is covered then
            $u' \leftarrow$ matched vertex of $v$
            $T \leftarrow T + uv + u'v$
            $X = X \cup \{u'\}$
            $Y = Y \cup \{v\}$
        end
    until $v$ is uncovered
    $P \leftarrow$ Path($u_0, v, T$): $M$-augmenting path
    $M \leftarrow M \triangle E(P)$
end
```

3 Matchings and vertex covers

In a graph G, we denote $\tau(G)$ the size of a minimum vertex cover of G, that is the minimum size of a vertex subset $X \subseteq V(G)$ such that every edge $e \in E(G)$ has an extremity in X. The parameters $\tau(G)$ and $\nu(G)$ are strongly correlated (they are dual of each other; although we will not focus on the specific meaning of that statement in this course). For bipartite graphs, that correlation translates to an equality.

Theorem 10 (Kőnig). *For every bipartite graph G,*

$$\nu(G) = \tau(G).$$

Proof. Let $G = (X, Y, E)$ be a bipartite graph, and let M be a maximum matching of G. We let $U := X \setminus V(M)$ be the set of vertices in U that are uncovered by M, and we let R be the set of vertices (including U) in G that can be reached from U with an M-alternating path. Since M is maximum, there is no M-augmenting path in G. This implies that every edge $e \in E$ either has both its extremities in R, or none.

Let $S := (Y \cap R) \cup (X \setminus R)$, then S is a vertex cover of G. Indeed, let $e = xy \in E$. Either both x and y are in R and hence e is covered by $Y \cap R$, or none are in R and hence e is covered by $X \setminus R$.

Since there is no M-augmenting path, $Y \cap R$ is saturated by M. Moreover, by definition, $X \setminus R$ contains only vertices that are matched by M to vertices not in R. We conclude that $|S| \leq |M|$, and hence $\tau(G) \leq \nu(G)$.

On the other hand, $\tau(G) \geq \nu(G)$ since given any vertex cover S, a vertex $s \in S$ covers at most one edge of M. This concludes the proof. $\square$

We note that there exists an equivalent formulation of Kőnig's theorem (the equivalence is proved in [TD4, Exercise 3]).

Theorem 11 (Kőnig, 2nd formulation)**.** *Let H be a bipartite graph on n vertices, and $\nu(H)$ the size of a maximum matching of H. Then*

$$\alpha(H) = n - \nu(H).$$

We also note that Hall's Theorem can be directly deduced from Kőnig's Theorem.

Second proof of Hall's Theorem. Let us assume that $G = (U, V, E)$ has no matching saturating U. Then by Kőnig's Theorem, G has a vertex cover S of size less than $|U|$. Let $X := U \setminus S$. Since S is a vertex cover, there is no edge between X and $V \setminus S$, hence $N(X) \subseteq V \cap S$. So

$$\begin{aligned} |N(X)| &\leq |S \cap V| = |S| - |S \cap U| \\ &< |U| - |U \cap S| = |U \setminus S| = |X|. \end{aligned}$$

We have found a set X that contradicts Hall's condition. This concludes the proof. □

4 Uses of matchings

4.1 Edge colouring

A proper *k-edge-colouring* of a given graph G can be equivalently defined as

- a proper k-colouring of its line-graph $L(G)$;
- a partition of $E(G)$ into k matchings;
- a mapping $c\colon E(G) \to [k]$ such that $c(e) \neq c(e')$ for every pair of incident edges e, e'.

The chromatic index $\chi'(G)$ is the minimum k such that there exists a proper k-edge-colouring of G. So we have $\chi'(G) = \chi(L(G))$.

In order to know a bit more on $\chi'(G)$, we can analyse the properties of the line-graph $L(G)$. The maximum degree $\Delta(L(G))$ is the maximum number of edges incident to a given edge $e = uv \in E(G)$ in G. There are at most $\Delta(G)$ edges incident to each extremity u and v, including e (twice). So the number of edges incident to either u or v is at most $2\Delta(G) - 2$ if we exclude e. We conclude that $\Delta(L(G)) \leq 2\Delta(G) - 2$. The clique number $\omega(L(G))$ is the size of a maximum set of edges that are pairwise incident. If we try to construct such a set, we can begin with two incident edges uv and uw. If we want to add a third edge e, we have two choices: either e is incident to u (this forms a star with 3 branches), either e is incident to both v and w (this forms a triangle). If we have a triangle, there is no way to add an edge that would be incident to all three edges of the triangle; any triangle is a maximal clique (in terms of inclusion) in $L(G)$. On the other hand, a star can be extended forever by adding edges incident to its central vertex. The largest star contained in G is the one whose central vertex has maximum degree $\Delta(G)$, hence $\omega(G) = \Delta(G)$, unless G has maximum degree 2 and contains a triangle, in which case $\omega(G) = 3$.

The usual bounds on the chromatic number gives us the following.

1. $\chi'(G) = \chi(L(G)) \leq \Delta(L(G)) + 1 \leq 2\Delta(G) - 1$.
2. $\chi'(G) = \chi(L(G)) \geq \omega(L(G)) \geq \Delta(G)$.

We note that there is a multiplicative gap of 2 between the lower and the upper bounds on $\chi'(G)$, so a greedy colouring algorithm performed on $L(G)$ will return a 2-approximation of an optimal edge-colouring of G. That gap is actually much smaller!

Theorem 12 (Vizing)**.** *For every graph G,*

$$\chi'(G) \leq \Delta(G) + 1,$$

hence $\chi'(G) \in \{\Delta(G), \Delta(G) + 1\}$.

The proof of Vizing's Theorem is constructive, and provides a polynomial algorithm to construct a $(\Delta(G)+1)$-edge-colouring of any graph G. However, deciding which of those two possible values is the correct one is an NP-complete problem for general graphs.

We observe that when G is Δ-regular, if $\chi(G) = \Delta$, then given a proper Δ-edge-colouring c of G, every vertex is incident with an edge of each colour in c. Said otherwise, the matching induced by each colour class covers every vertex of G; this is a perfect matching. So $\chi'(G) = \Delta$ if and only if $E(G)$ can be decomposed into perfect matchings. This is in particular always the case if G is bipartite [TD4, Question 1.2].

4.2 Factors

Given a graph G, a k-factor of G is a spanning k-regular subgraph of G. In particular, a 1-factor of G is a perfect matching of G, and a 2-factor of G is a union of cycles that cover $V(G)$.

We note that the union of k perfect matchings yields a k-factor, while the reverse is false in general. For instance, a Hamiltonian graph with an odd number of vertices contains a 2-factor (an Hamiltonian cycle), but it contains no perfect matching (a matching can only cover an even number of vertices).

In order to deduce properties on edge colourings, it it interesting to be able to decompose the edge set of a given graph into k-factors for small values of k (if we can decompose into 1-factors, then we have an optimal edge-colouring). It turns out that we can always do it with $k = 2$ if the graph is regular and of even degree.

Theorem 13 (Petersen). *For every $r > 0$, every $2r$-regular graph can be decomposed into r 2-factors.*

The Theorem of Petersen relies on that of Euler, which states that a graph is Eulerian if and only if all its vertices have even degrees. A graph G is Eulerian if it contains an Eulerian circuit, that is a circular ordering of $E(G)$ such that two consecutive edges are incident. Equivalently, G is Eulerian if and only if its line graph $L(G)$ is Hamiltonian.

Proof. We know by Euler's theorem that a graph with only even degrees has a Eulerian circuit. A traversal of this Eulerian circuit yields an orientation $\overrightarrow{G}$ of the edges of G, such that every vertex has r in-going arcs and r out-going arcs. Let $H = (V^-, V^+, E_H)$ be the r-regular bipartite graph obtained by duplicating each vertex $v \in V(G)$ into $v^- \in V^-$ and $v^+ \in V^+$, and putting an edge between u^- and v^+ whenever there is an arc going from u to v in $\overrightarrow{G}$.

Since H is bipartite and regular, it can be decomposed into $2r$ perfect matchings. It is possible to organise them into r pairs, so that each pair of matchings yields a 2-factor of G after collapsing each pair of vertices $(v^-, v^+) \in V^- \times V^+$ back into $v \in V(G)$. $\square$

Let us make an exception and consider multigraphs. In a multigraph G, there might be several edges between the same pair of vertices. The Theorem of Petersen holds also for multigraphs, and this has an interesting consequence on the chromatic index of multigraphs.

Let G be a $2r$-regular multigraph, and let $F_1, \dots, F_r$ be a decomposition of $E(G)$ into 2-factors. Each F_i is a disjoint union of cycles (some of which might be of length 2), and is therefore 3-edge-colourable. We introduce 3 distinct colours for each F_i, and properly colour the edges of F_i with those colours; this yields a proper 3-edge-colouring c_i of F_i. Because the sets of colours used by two distinct colourings c_i, c_j are disjoint, their union creates no conflict. Therefore, the union of the colourings c_i yields a proper $3r$-edge-colouring of G.

More generally, we can deduce from Petersen's Theorem the extremal value of the chromatic index of multigraphs in terms of their maximum degree.

Theorem 14 (Shannon). *For every multigraph G,*

$$\chi'(G) \leq \left\lfloor \frac{3\Delta(G)}{2} \right\rfloor,$$

and this is attained by Shannon's triangles.

For simple graphs, there exists an even tighter bound.

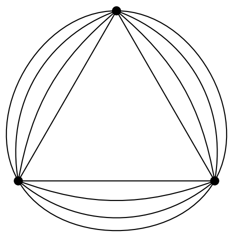

Figure 4.1: Shannon’s Theorem is sharp for Shannon’s triangles.