Algorithms

— To get a taste of graph algorithms — it seems a good idea to have a look at the early achievements that are still widely in use today.

Before we start a word of warning.[1] Debugging may help you find errors in your algorithm, but it can never prove that your algorithm is correct.

[1] I read this in one of Dijkstra's very early articles.

— Actually[2] — there cannot be a procedure that tests if an algorithm terminates or not. Minsky gives the following proof of this. Assume there were a procedure proper(·) that takes as input any procedure L and outputs TRUE if L terminates and FALSE otherwise. The following example provides a contradiction.

[2] In Concrete Math the side note "skip to the next section" would appear here.

Consider the procedure L shown in Algorithm 2. When L is proper then it is improper and vice versa. Hence, the procedure proper(·) can not exist.

```
1: procedure L
2:   use package proper
3:
4:     while proper(L) do
5:         whistle once
6:     end while
7: end procedure
```

Algorithm 2: Minsky's example of a procedure that is neither proper nor improper.

T. Kloks, M. Xiao, *A Guide to Graph Algorithms*, https://doi.org/10.1007/978-981-16-6350-5_2

2.1 Finding and counting small induced subgraphs

TRIANGLES IN GRAPHS can be found via fast matrix multiplication of its adjacency matrix. This gives an algorithm to find a triangle in $O(n^\alpha)$ where $\alpha < 2.376$.

In 1997 Alon, Yuster and Zwick showed that a triangle can be found in $O(m^{2\alpha/\alpha+1}) = O(m^{1.41})$. (This improved an earlier $O(m^{3/2})$ algorithm by Itai and Rodeh.) [3]

In this section we show how their method can be used to find a diamond in a graph in $O(m^{3/2} + n^\alpha)$. [4]

[3] In this section we express the run - time of algorithms as a function of n and m.

[4] A diamond is a graph with 4 vertices; obtained from K_4 by removing one edge. A graph is diamond-free if no induced subgraph is isomorphic to the diamond.

Exercise 2.1

A graph is diamond - free if and only if the neighborhood of every vertex induces a graph in which every component is a clique.

Hint: If a graph has no diamond then every neighborhood is P_3-free — ie — every component of a neighborhood is a clique.

The algorithm to check if a graph G has a diamond first partitions the vertices in those that have 'low' degree and 'high' degree. Let D be some number. A vertex is low degree if its degree is at most D and otherwise it is high degree. Let L be the set of vertices that have low degree and let H be the set of vertices that have high degree.

The search for a diamond is split in 4 parts.

Phase 1. check if G has a diamond with a vertex of degree 3 that is of low degree

Phase 2. check if G has a diamond with a vertex of degree 2 that is of low degree

Phase 3. if no diamond was found then remove all vertices of low degree. Let G^* be the graph that remains.

Phase 4. check if G^* has a diamond.

We show how each phase is implemented. We assume that the adjacency matrix A is given. For each $x \in L$ construct adjacency lists for the graph induced by $N(x)$. This can be accomplished in $O(d(x)^2)$ time. Then compute the components of $G[N(x)]$ and check if each component is a clique. This can be done in $O(d(x)^2)$ time. If some component is not a clique then a P_3 is found in $O(d(x)^2)$ time. [5] It follows that this phase can be completed in time

[5] Exercise !

$$\sum_{x \in L} d(x)^2 \quad \leqslant \quad 2 \cdot D \cdot m.$$

We describe the implementation of Phase 2. Let $x \in L$ and let C be a maximal clique in $N(x)$. For each pair $y, z \in C$ check if

$$A^2_{y,z} \quad > \quad |C| - 1.$$

If that is the case then y and z have a common neighbor outside $N[x]$ — ie — we find a diamond. The diamond can be produced in linear time when all adjacency lists are sorted. We leave it as an exercise to check that Phase 2 runs in $O(D \cdot m + n^\alpha)$.

Assume that Phase 1 and Phase 2 do not produce a diamond. Notice that

$$V(G^*) \quad \leqslant \quad \frac{2 \cdot m}{D}.$$

Repeat the procedure described in Phase 1 for all vertices of G^*. This can be implemented to run in

$$\sum_{x \in H} d_H(x)^2 \quad = \quad O\left(m \cdot |H|\right) \quad = \quad O\left(\frac{m^2}{D}\right).$$

Theorem 2.1. *There exists an algorithm that finds a diamond in a graph if there is one. With the adjacency matrix of the graph as input the algorithm runs in* $O(n^\alpha + m^{3/2})$.

Proof. By the above the total run - time is at most

$$O\left(D \cdot m + \frac{m^2}{D} + n^\alpha\right).$$

Choose $D = \sqrt{m}$. □

Exercise 2.2

Use a similar technique to show there is an algorithm to check if a connected graph is claw-free that runs in $O(m^{(\alpha+1)/2}) = O(m^{1.69})$. [6]

[6] The claw is shown in Figure 2.11 on Page 84. It is a tree with 4 vertices of which 3 are leaves.

HINT: If a graph is claw-free then every vertex has at most $2\sqrt{m}$ neighbors. When every vertex has at most $2\sqrt{m}$ neighbors then do a fast matrix multiplication for each neighborhood and check for a $\bar{K}_3$. This step can be performed in time proportional to

$$\sum_x d(x)^\alpha \quad \leqslant \quad (2\sqrt{m})^{\alpha-1} \cdot \sum_x d(x) \quad \leqslant \quad 2^\alpha \cdot m^{(\alpha+1)2}.$$

2.2 Bottleneck domination

Let G be a graph and let $w : V \to \mathbb{R}$ be a function which assigns to every vertex x a weight $w(x)$. We assume that arithmetic operations on vertex weights can be performed in $O(1)$ time.

Definition 2.2. Let (G, w) be a weighted graph. For $W \subseteq V$ the bottleneck of W is

$$\max\{w(x) \mid x \in V\}.$$

Definition 2.3. Let G be a graph. A set $D \subseteq V$ is a dominating set if every vertex of $V \setminus D$ has a neighbor in D.

For example, every maximal independent set in a graph is a dominating set.

The bottleneck domination problem is the following.
Input: A weighted graph (G, w).
Output: A dominating set with minimal bottleneck.

T. Kloks, D. Kratsch, C. Lee and J. Liu, *Improved bottleneck domination algorithms*, Discrete Applied Mathematics **154** (2006), pp. 1578–1592.

Exercise 2.3

Prove the following theorem.

Theorem 2.4. *There exists a linear time - algorithm to solve the bottleneck domination problem.*

HINT: Let (G, w) be a weighted graph. For $x \in V$ let

$$m(x) \quad = \quad \min\{w(y) \mid y \in N[x]\}.$$

Let

$$\rho \quad = \quad \max\{m(x) \mid x \in V\}.$$

Show that the minimal bottleneck is ρ.

Definition 2.5. Let G be a graph. A total dominating set is a set $D \subseteq V$ such that every vertex of V has a neighbor in D. [7]

[7] If the graph has isolated vertices then there is no total dominating set.

Exercise 2.4

Show that D is a total dominating set if it is a dominating set and $G[D]$ has no isolated vertices.

Exercise 2.5

Prove the following theorem.

Theorem 2.6. *There exists a linear time - algorithm to compute a total dominating set with smallest bottleneck in a weighted graph.*

HINT: For $x \in V$ define

$$m'(x) \quad = \quad \min\{w(y) \mid y \in N(x)\}$$

and let

$$\rho' \quad = \quad \max\{m'(x) \mid x \in V\}.$$

Show that the smallest bottleneck of a total dominating set is ρ'.

2.3 The Bron & Kerbosch Algorithm

In this section we'll have a look at the Bron – Kerbosch algorithm. It was developed in the early 1970s at Eindhoven's University of Technology in The Netherlands. It remains to this day a winner.

Recall the definition of a clique (Definition 1.21).[8]

[8] A clique in a graph G is a set of vertices that are all pairwise adjacent.

Definition 2.7. A clique C is maximal if

$$\forall_{x \notin C} \exists_{y \in C} \quad \{x, y\} \notin E. \tag{2.1}$$

For a graph G let

$$\Omega(G)$$

represent the set of maximal cliques in G.

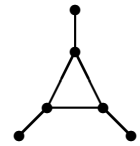


Figure 2.1: A clique is maximal if it is not contained in a larger one. Obviously, this doesn't mean that there is no larger one! This graph is called the net. Edges incident with pendant vertices are maximal cliques. The largest maximal clique is the triangle. A largest maximal clique is called a maximum clique.

The algorithm of Bron and Kerbosch lists all the maximal cliques in a graph. In this section we describe a variant of the original Bron and Kerbosch – algorithm and we analyze its time complexity.

When a graph G has only vertices of degree at most 2 then each component of G is a path or a cycle. In that case the number of maximal cliques is at most n. It seems that when a graph has a lot of maximal cliques it has a lot of vertices of high degree.

Let's look at the case where all vertices of G have degree at least $n-3$. We say that G is high degree.

Exercise 2.6

Show that a graph G is high degree if and only if every induced subgraph of G is that.

Figure 2.2: This figure show the complement of a high degree-graph. For this graph we use notations

$$K_3 + \cdots + K_3 = t \cdot K_3,$$

to be read as 'a union of triangles.'

Notice that, when G is high degree, every component of $\bar{G}$ is a path or a cycle. Consider the case where $\bar{G}$ is a path — say

$$[x_1 \quad \cdots \quad x_n].$$

We can list the maximal cliques of G as follows. When $n \leqslant 3$ this is easy — so we assume henceforth that $n \geqslant 4$.

For the set of maximal cliques in $\bar{P}_n$ we have

$$\Omega(1 \cdots 1) = \{\{x_1\}\}, \quad \Omega(1 \cdots 2) = \{\{x_1\}, \{x_2\}\}, \quad \text{and}$$
$$\Omega(1 \cdots 3) = \{\{x_1, x_3\}, \{x_2\}\}.$$

and for $n \geqslant 4$,

$$\Omega(1 \cdots n) = \{\{x_n\} \cup C \mid C \in \Omega(1 \cdots n-2)\} \bigcup \{\{x_{n-1}\} \cup C \mid C \in \Omega(1 \cdots n-3)\}. \quad (2.2)$$

Exercise 2.7

Implement this algorithm to list all maximal cliques in the complement of a path.

For the number of maximal cliques in $\bar{P}_n$ we have the recurrence

$$P(1) = 1 \quad P(2) = 2 \quad P(3) = 2 \quad \text{and}$$
$$P(n) = P(n-2) + P(n-3) \quad \text{for } n \geqslant 4,$$

where $P(n) = |\Omega(\bar{P}_n)|$. This solves as $O\left(\left(\frac{4}{3}\right)^n\right)$. (The architect Dom van der Laan considered this the ideal fraction of measurements for his buildings.)

Exercise 2.8

Design a similar algorithm to list all the maximal cliques in the complement of a cycle.
Hint: Can you adapt the algorithm for paths?

Some straightforward calculations show that the high-degree graph — with the most maximal cliques — is a graph G — satisfying one of the following.

i. If $n = 0 \bmod 3$, all components of $\bar{G}$ are triangles, that is, G has $3^{n/3}$ maximal cliques.

ii. If $n = 1 \bmod 3$ and $n > 1$, all components of $\bar{G}$, except two, are triangles, and the two exceptional components are edges.[9] This gives $4 \cdot 3^{(n-4)/3}$ maximal cliques if $n > 1$. We have one maximal clique if $n = 1$.

iii. If $n = 2 \bmod 3$, all components of $\bar{G}$ — except one — are triangles, and the exceptional component is a single edge. This case gives $2 \cdot 3^{(n-2)/3}$ maximal cliques.

It follows that every high degree-graph has at most $3^{n/3}$ maximal cliques.

Let us first show that — indeed — among all graphs these graphs are the ones with the most maximal cliques.

Lemma 2.8. *Let G be a graph which is not high degree. Then*

$$|\Omega| \leqslant \mu \cdot 3^{n/3} \tag{2.3}$$

for some $\mu \in \mathbb{R}$, $0 < \mu < 1$. Here $n = |V(G)|$.

Proof. The graph must have a vertex x of degree at most $n - 4$. Partition the set of maximal cliques into those that contain x and those that do not contain x. The first set of cliques are exactly the maximal cliques contained in $G[N(x)]$, with x added on to each of them.[10] The second set of cliques are maximal cliques in $G - x$. Possibly this second set contains

[9] The two exceptional components induce the complement of a 4-cycle with 4 maximal cliques.

Exercise 2.9

Part I
Consider the high degree-graph G in Figure 2.2. This graph is the complement of $\frac{n}{3}$ triangles. Show that

$$|\Omega(G)| = \left(3^{1/3}\right)^n = (1.442\cdots)^n.$$

Part II
Show that the graph above has the most maximal cliques among all high degree-graphs.

[10] If $N(x) = \varnothing$, the vertex x is an isolated vertex. In that case the only maximal clique that contains x is $\{x\}$.

cliques that are not maximal in G — but — as an upperbound we get

$$|\Omega(G)| \leqslant \begin{cases} |\Omega(G[N(x)])| + |\Omega(G-x)| & \text{If } N(x) \neq \varnothing \\ 1 + |\Omega(G-x)| & \text{otherwise.} \end{cases} \tag{2.4}$$

By induction on the number of vertices in the graph

$|\Omega(G[N(x)])| \leqslant 3^{(n-4)/3}$ since x has degree at most $n-4$.

By induction also

$$|\Omega(G-x)| \leqslant 3^{(n-1)/3}.$$

It follows that [11]

$$\begin{aligned} |\Omega(G)| &\leqslant 3^{(n-4)/3} + 3^{(n-1)/3} \\ &= (3^{-4/3} + 3^{-1/3}) \cdot 3^{n/3} \\ &= 3^{-4/3} \cdot (1+3) \cdot 3^{n/3} \\ &= 3^{-4/3} \cdot 4 \cdot 3^{n/3} \\ &< 3^{n/3} \quad \text{since } 3^{4/3} > 4.326. \end{aligned}$$

This completes the proof. □

[11] Since G is not high degree we may assume $n \geqslant 4$. When x is isolated, we have, for some $\mu \in (0,1)$, since $n \geqslant 4$,

$$\begin{aligned} |\Omega| &\leqslant 1 + 3^{(n-1)/3} \\ &\leqslant \mu \cdot 3^{n/3}. \end{aligned}$$

Remark 2.9. I agree to it that the notation in (2.4) is *awful* with all those (useless) brackets! One of my teachers, Professor De Bruijn used to say: " Our notation for functions is terrible. — Unfortunately — it's not bad enough for people to feel the need to change it; so we're stuck with it and we'd better get used to it ! "

Lemma 2.10. *There is an algorithm that lists all the maximal cliques in a high degree graph in* $O(n^2 \cdot |\Omega|)$ *time.*

Proof. There is an algorithm that runs in $O(n^2)$ time and that computes the components of $\bar{G}$. [12] Since G is high degree,

[12] We assume that G is suitably represented.

each component of $\bar{G}$ is a path or a cycle, and we can find a suitable vertex ordering in each of these components.

The technique — explained on Page 23 ff — shows that there is an algorithm, that lists all cliques in the complement of a path or cycle in time $O(|W| \cdot N)$, where N is the number of maximal cliques in $G[W]$ for a component W of $\bar{G}$.

Let $\{W_1, \cdots, W_k\}$ be the set of components of $\bar{G}$. Then

$$\Omega(G) = \{ S_1 \cup \cdots \cup S_k \mid \forall_i \quad S_i \text{ is a clique in the component } G[W_i] \}. \quad (2.5)$$

It is now straightforward to show that all maximal cliques in G can be listed in $O\left(n^2 \cdot |\Omega(G)|\right)$ time. □

Exercise 2.10

Implement and run the algorithm described above for C_5.

Let A be the algorithm described above, that lists all maximal cliques in graphs that are high degree in

$$O\left(n^2 \cdot |\Omega|\right) = O\left(n^2 \cdot 3^{n/3}\right) \quad \text{time (as } n \to \infty).$$

When we estimate time-bounds of graph algorithms, we are usually interested in the case where $n \to \infty$. We usually omit this addendum.

Choose=pick=select

We describe the algorithm of Bron and Kerbosch blow — in Algorithm 3. The algorithm to list all the maximal cliques consists of a call to the procedure with parameters:

$$B\&K(\varnothing, V, \varnothing).$$

An invariant is a property that holds true for the parameters of a procedure, at every call to it. An invariant is useful when the termination condition together with the invariant yields the desired solution, ie, the 'post condition.' The concept of an invariant of a procedure was introduced by Dijkstra in the early 1960s. As a concept it is a useful tool to prove the correctness of programs.

In the case of the procedure $B\&K$ described above the parameters can be described by the following invariant:

Algorithm 3: The Bron – Kerbosch Algorithm

```
 1: procedure B & K( R, P, X )
 2:     if P ∪ X = ∅ then report R
 3:     else
 4:         if P = ∅ then skip
 5:         else
 6:             if G[P] is high degree then
 7:                 Compute Ω( G[P] ) using Algorithm A and
 8:                 extend them with R
 9:             else
10:                 Choose x ∈ P such that |N(x) ∩ P| < |P| − 4
11:                 B & K( R ∪ {x}, P ∩ N(x), X ∩ N(x) )
12:                 B & K( R, P \ {x}, X ∪ {x} )
13:             end if
14:         end if
15:     end if
16: end procedure
```

1. $R = \varnothing$ or R is a clique in G.

2. P and X are disjoint sets and

$$P \cup X = \{\, y \mid R \subseteq N(y) \,\}$$

— that is — the set $P \cup X$ contains those vertices $y \in V \setminus R$ such that $\{y\} \cup R$ is a clique.

Notice that — by virtue of the invariant — the set R is a maximal clique exactly when $P \cup X = \varnothing$; thence the Report command in Line 2.

The set P is called the set of candidates. When x is a candidate, chosen in Line 10, the algorithm lists all maximal cliques that contain $R \cup \{x\}$. Then x is removed from the set of candidates and put into the set X to maintain the invariant. — Finally — the remaining set of maximal cliques, that is, those that do not contain x, are listed via the call B & K with parameters

$$R, \quad P \setminus \{x\}, \quad \text{and} \quad X \cup \{x\}.$$

Exercise 2.11

Implement and run the Bron – Kerbosch algorithm for the 5-wheel W_5 in Figure 2.3 and for the Petersen graph (Figure 1.2 on Page 3).

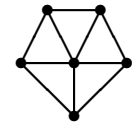


Figure 2.3: The 5-wheel W_5

2.3.1 A Timebound for the B & K – Algorithm

We have seen that when G is high – degree, its maximal cliques can be listed using algorithm A in $O\left(n^2 \cdot 3^{n/3}\right)$ time. The sets P, X and R in the Bron and Kerbosch – algorithm may be implemented by a pointer structure or an array.

Theorem 2.11. *The Bron – Kerbosch algorithm runs in*

$$O(n^2 \cdot 3^{n/3}) \quad \textit{time}.$$

Proof. Let $t(n)$ denote the time needed to list all maximal cliques in a graph with n vertices. Write

$$t(n) = t^*(n) + r(n),$$

where $r(n)$ is the time spent on reporting maximal cliques. Then

$$r(n) = O(n \cdot |\Omega|).$$

The variant of the Bron – Kerbosch algorithm, in which each report statement is replaced by an $O(1)$ statement — like increasing a counter to count the number of maximal cliques — has then running time $t^*(n)$.

When $G[P]$ is high degree we have, by Lemma 2.10,

$$t^*(p) = O\left(3^{p/3}\right) \quad \text{where } p = |P|. \tag{2.6}$$

We claim that — when $G[P]$ is not high degree

$$t^*(p) \leqslant t^*(p-4) + t^*(p-1) + O\left(p^2\right). \tag{2.7}$$

The term $t^*(p-4)$ corresponds to the case where a candidate x of degree at most $p-4$ is added to R. The search space reduces to $N(x)$, which has at most $p-4$ vertices.

The term $t^*(p-1)$ is the time needed to count the maximal cliques that do not contain x.[13] — In that case — the search space is $P-x$, ie, a graph with $p-1$ vertices.

[13] That is, there must exist a vertex $y \in P \setminus N_P[x]$ that is adjacent to all vertices of R.

The term $O(p^2)$ is the time needed to check if $G[P]$ is high degree, to update the sets R, P and X, and to find a candidate $x \in P$ with at most $p-4$ neighbors in P.

The recurrence (2.7) is similar to what we obtained for the number of maximal cliques in 2.4 on Page 25. It is readily checked that

$$t^*(n) = O\left(n^2 \cdot 3^{n/3}\right).$$

IMPORTANT: Make sure that you know how to solve a recurrence as in (2.7). If not, ask your teacher or check-out Concrete Math!

This completes the proof. □

Remark 2.12. After having gathered a list of all the maximal cliques in the graph $G[N[x]]$, the remaining maximal cliques must contain at least one nonneighbor of x. — So — instead of finding all maximal cliques in $G-x$, we could list all maximal cliques that contain some nonneighbor of x. The analysis of Tomita et al. shows that this is 'much of muchness' — it does not improve the worst-case time estimate.

Exercise 2.12

This idea can be built into the algorithm by replacing the second recursive call (in Line 12) by a loop during which all nonneighbors of x are tried. In order to minimize the number of recursive calls Tomita et al. choose a vertex x with the most neighbors.

Eppstein et al. analyze the complexity for sparse graphs in terms of their 'degeneracy.' [14]

[14] A graph is k-degenerate if every induced subgraph has a vertex of degree at most k.

Remark 2.13. For the maximal number of maximal cliques that a graph with $n > 1$ vertices may have, Moon and Moser derived the following formula.

$$g(n) = \begin{cases} 3^{n/3} & \text{if } n = 0 \bmod 3 \\ 4 \cdot 3^{\lfloor n/3 \rfloor - 1} & \text{if } n = 1 \bmod 3 \\ 2 \cdot 3^{\lfloor n/3 \rfloor} & \text{if } n = 2 \bmod 3\,. \end{cases} \tag{2.8}$$

For a slightly different proof see also Vatter.

The graph in Figure 2.2 is the unique[15] graph with n vertices and $3^{n/3}$ maximal cliques (when $n = 0 \bmod 3$).

[15] up to isomorphism

2.4 Total Order!

A binary relation on a set P is a subset of the set of ordered pairs in P, ie, a subset of the Cartesian product $P^2 = P \times P$ where

$$P \times P = \{ (q, r) \mid q \in P \quad \text{and} \quad r \in P \}.$$

When P is a poset, we use the same symbol P also for its set of elements. This abuse of notation was also common in graph theory. De Bruijn used to say about this: "As long as you know what you're talking about, there's no problem at all!"

Definition 2.14. A partial order — or poset — P is a pair

$$(P, \leqslant),$$

where P is a set and $\leqslant$ is a binary relation on P satisfying:

$$\begin{array}{rlcll} \forall_{x \in P} & x \leqslant x & & & \leqslant \text{ is reflexive,} \\ \forall_{x \in P}\, \forall_{y \in P} & (x \leqslant y \;\&\; y \leqslant x) & \Rightarrow & x = y & \text{antisymmetric,} \\ \forall_{x \in P}\, \forall_{y \in P}\, \forall_{z \in P} & (x \leqslant y \;\&\; y \leqslant z) & \Rightarrow & x \leqslant z & \text{and transitive.} \end{array}$$

Notice that possibly there are elements $x \in P$ and $y \in P$ for which neither $x \leqslant y$ nor $y \leqslant x$ holds.

Definition 2.15. A partial order $(P, \leqslant)$ is a total order if every pair of its elements are related by $\leqslant$. [16]

[16] For example, $[n]$ and $\mathbb{N}$ are total orders. A total order is also called a linear order.

The TOTAL ORDERING PROBLEM is the problem to find a total order of a set V satisfying a collection of 'betweenness constraints.' We are given a collection R of ordered triples

'constraint' = 'restriction.'

$$(a, b, c) \in V^3.$$

The total order $\leqslant$ should satisfy:

$$(a, b, c) \in R \quad \Rightarrow \quad (a < b < c) \quad \text{or} \quad (c < b < a),$$
$$\text{where we use } p < q \text{ to denote that } p \leqslant q \text{ and } p \neq q. \tag{2.9}$$

Paraphrased — the required total order $\leqslant$ puts b 'between' a and c.

Let's look at the SIMPLE TOTAL ORDERING PROBLEM first. That problem is similar to the one above — except that each betweenness constraint has the form

$$(a, b, c) \in R \quad \Rightarrow \quad a < b < c. \tag{2.10}$$

In the simple total ordering problem the collection of constraints builds a poset on the elements of V. In other words, the betweenness constraints define arcs between elements of V, say $x \to y$ if some betweenness relation implies that $x < y$.

A digraph is a graph in which each edge $\{x, y\}$ has a direction, either $x \to y$ or $y \to x$. A digraph is not a graph! If a digraph has no directed cycles, it is called a DAG, a directed, acyclic graph.

Exercise 2.13

A simple total ordering problem on V has a solution if and only if the directed graph — defined above — is a DAG.

A topological sort of a digraph is a total ordering of its vertices such that for every arc $x \to y$ the vertex x comes before y in the total order.

In 1962 Kahn described an algorithm that finds a topological sort in a DAG — say $G = (V, A)$ — in time $O(n + m)$. [17]

[17] $n = |V|$ and $m = |A|$

In Kahn's algorithm, Algorithm 4 on Page 32, the set S is the set of 'start–nodes,' which is the set of vertices without incoming arcs.[18] The set L contains the final linear ordering.

[18] Start-nodes are 'sources.' The vertices without outgoing arcs are called 'sinks.'

Algorithm 4: Kahn's Topological Sort

```
1:  procedure KAHN( G = ( V , A ) )
2:
3:      L ← ∅
4:      S ← { x ∈ V | ∀_{y∈V} ¬( y → x ) }   ▷ S is the set of sources.
5:
6:      while S ≠ ∅ do
7:          x ← ∈ S
8:          S ← S \ { x }
9:          L ← L + { x }                    ▷ x is added at the end of L.
10:
11:         for (x, y) ∈ A do
12:             A ← A \ { (x, y) }
13:             if ∀_z ¬( (z, y) ∈ A ) then      ▷ y is a new source.
14:                 S ← S ∪ { y }
15:             end if
16:         end for
17:
18:     end while
19:
20:     if A ≠ ∅ then
21:         G has a cycle: Report defeat
22:     else
23:         Report L
24:     end if
25:
26: end procedure
```

Exercise 2.14

Prove that any start–node may start a topological sort of a DAG.

A start – node x is selected as a first element in L. The vertex x is then removed from the graph and a topological sort is performed on the remaining graph. In the remaining graph, all arcs (x, y) are removed — since x is no longer a vertex of the graph — and the set of start – nodes of $G - x$ is determined. The process continues as long as the set of start – nodes is nonempty.

Exercise 2.15

Prove that upon completion of the algorithm described above — that is when $S = \varnothing$ — any remaining arc implies that G has a cycle.
Hint: The remaining digraph — if any — has no source.

Theorem 2.16. *There exists a linear – time algorithm* [19] *that computes a topological sort in a* DAG.

[19] By 'linear' we mean that the algorithm runs in $O(n + m)$ time.

Proof. The graph is represented as a list of arcs. For each vertex the algorithm maintains a list of out – neighbors and a list of in – neighbors. These lists are updated in $O(1)$ time whenever an arc is removed.

To find the set of start – nodes, define a Boolean array

$$b : V \to \{\text{TRUE}, \text{FALSE}\}$$

and initialize it as TRUE for all vertices. Then the algorithm passes through all arcs $(x, y) \in A$ and sets $b : (y) = \text{FALSE}$. The start – nodes are then the remaining vertices, ie, those x for which $b(x)$ remained TRUE. This part of the algorithm can be implemented so that it runs in $O(n + m)$ time.

At each pass of the loop in Kahn's algorithm, starting at Line 7, a vertex x is removed from S and added to L. Effectively, x is removed from the graph. The removal of arcs that leave x takes $O(1)$ time per arc. The fact that each arc is removed at most once proves the timebound.

This proves the theorem. □

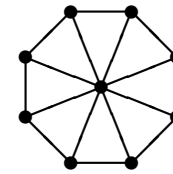


Figure 2.4: The 8-wheel W_8 (The figure appears here for no particular reason.)

— By Theorem 2.16 — there exists a linear–time algorithm that solves the simple total ordering problem. Unfortunately — there is no method like that available for the total ordering problem.

Opatnrý shows that the problem to color a hypergraph of rank 3 can be reduced to the total ordering problem. We explore this issue in the next section.

2.4.1 Hypergraphs

Definition 2.17. A hypergraph H is a pair $(V, \mathcal{E})$ — where V is a finite nonempty set and $\mathcal{E} = E(H)$ is a set of nonempty subsets of V. [20]

The elements of $\mathcal{E}$ are called hyperedges.

[20] Hypergraphs are sets of subsets of a 'universal set' of vertices V.

Definition 2.18. The rank of a hypergraph H is the maximal cardinality of its hyperedges, ie,

$$\text{rank}(H) = \max\{|e| \mid e \in \mathcal{E}\},$$
$$\text{where } \mathcal{E} = E(H) \text{ denotes the set of hyperedges of } H. \quad (2.11)$$

A hypergraph is called k-uniform if all its hyperedges have k vertices.

The 2–coloring problem for hypergraphs $H = (V, \mathcal{E})$ is to find a partition of its vertices — say $\{A, B\}$ — such that each hyperedge has a nonempty intersection with A and B, ie,

$$\forall_{e \in \mathcal{E}} \quad e \cap A \neq \varnothing \quad \text{and} \quad e \cap B \neq \varnothing. \quad (2.12)$$

When H is 2–uniform then H is a graph and then the 2–coloring problem is easy to solve. [21]

[21] See exercise 1.14.

In the remainder of this section we show that the total ordering problem is at least as hard as the 2–coloring problem for hypergraphs of rank 3. By that we mean that if there exists a polynomial–time algorithm that solves the total ordering problem, then that algorithm can be adapted, so that it solves the 2–coloring problem for hypergraphs of rank 3 in polynomial time.

Exercise 2.16

An orientation of a graph gives each edge a direction. Show that any graph can be oriented into a DAG.

Exercise 2.17

Orient the graph in Figure 2.4 into a DAG. Implement Kahn's algorithm, and find a topological sort.

Remark 2.19. Cook showed that the 2-coloring problem for hypergraphs of rank 3 is NP-complete. There are some indications that show that there is — probably — no polynomial-time algorithm for any NP-complete problem. We will speak more about that in our chapter on complexity.

2.4.2 Problem Reductions

The method — to show that the total ordering problem is at least as hard as the 2-coloring problem — is called a problem reduction. We reduce the 2-coloring problem for hypergraphs of rank 3 to the total ordering problem and we show that this reduction takes polynomial time.

Assume that there exists a polynomial-time algorithm that solves the total ordering problem in time $O\big((n+|R|)^k\big)$, for some $k \in \mathbb{N}$. Here $n = |V|$, the cardinality of the universal set, and R is the set of betweenness constraints. We show that there also exists an algorithm that solves the 2-coloring problem for hypergraphs H of rank 3 in $O\Big((n+m)^k\Big)$ time. Here $n = |V(H)|$ and $m = |E(H)|$.

Proof. Let H be a hypergraph of rank 3 for which we wish to solve the 2-coloring problem. Notice that we may assume that there are no hyperedges of cardinality 1, otherwise there cannot exist a 2-coloring of H and we are done. Henceforth, we assume that every hyperedge is either a triple or a pair of elements of V.

Number the vertices and hyperedges of H:

$$\begin{aligned} V(H) &= \{h_1, \cdots, h_n\} \quad \text{and} \\ E(H) &= \{(a_i, b_i, c_i) \mid \{a_i, b_i, c_i\} \subseteq V \quad \text{and} \quad 1 \leqslant i \leqslant t\} \\ &\bigcup \{(d_j, e_j) \mid \{d_j, e_j\} \subseteq V \quad \text{and} \quad 1 \leqslant j \leqslant p\}. \end{aligned} \tag{2.13}$$

Construct a universal set and a set of betweenness constraints as follows. For each triple $(a_i, b_i, c_i) \in E(H)$ introduce one vertex, say y_i. Add one more vertex x. The universal set V^* is

$$V^* = V(H) \cup \{x\} \cup \{y_i \mid (a_i, b_i, c_i) \in E(H)\}. \tag{2.14}$$

The set of betweenness constraints R is defined as follows.

For each triple $(a_i, b_i, c_i) \in E(H)$ the following two triples are in R,

$$(a_i, y_i, b_i) \in R \quad \text{and} \quad (y_i, x, c_i) \in R. \tag{2.15}$$

For each pair $(d_i, e_i) \in E(H)$ the following triple is in R:

$$(d_i, x, e_i) \in R. \tag{2.16}$$

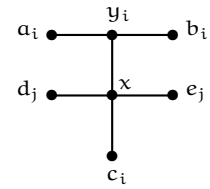


Figure 2.5: The figure illustrates the betweenness constraints.

This completes the description of the betweenness relations and the universal set.

We claim that H has a 2–coloring if and only if there is a total ordering of V^* satisfying the betweenness constraints R.

Assume that the hypergraph H has a 2–coloring of its vertices. Let $\{A, B\}$ be a partition of V that contains an endpoint of every hyperedge. We show that there is a linear ordering of V^* satisfying R.

Exercise 2.18

Show that the function f is injective.

Construct an injective map $f : V^* \to \mathbb{Q}$ as follows.

$$f(x) = 0 \tag{2.17}$$

$$\forall_{h_\ell \in V} \quad f(h_\ell) = \begin{cases} \ell & \text{if } h_\ell \in A \\ -\ell & \text{if } h_\ell \in B, \end{cases} \tag{2.18}$$

$$\forall_{i \in [t]} \quad f(y_i) = \begin{cases} \min\{f(a_i), f(b_i)\} + \frac{1}{i+1} & \text{if } \mathsf{sign}(f(a_i)) = \mathsf{sign}(f(b_i)) \\ -\frac{\mathsf{sign}(f(c_i))}{i+1} & \text{otherwise.} \end{cases} \tag{2.19}$$

$$\forall_{z \in \mathbb{Z}} \ \mathsf{sign}(z) = \begin{cases} 1 & \text{if } z > 0 \\ -1 & \text{if } z < 0. \end{cases}$$

Notice that each vertex of H is mapped to i or to $-i$, for some natural number $i \in \mathbb{N}$. It is positive when it is in

A and negative otherwise. Since the endpoints of each hyperedge $(c_i, d_i) \in E(H)$ have opposite colors each betweenness constraint $(c_i, x, d_i) \in R$ is satisfied.

Consider $(a_i, b_i, c_i) \in E(H)$. We check if the betweenness constraint (a_i, y_i, b_i) is satisfied.

If a_i and b_i are both in the same sets of the partition, ie, if

$$\mathsf{sign}(f(a_i)) = \mathsf{sign}(f(b_i))$$

then — since $f(a_i) \neq f(b_i)$ [22] — y_i is mapped to the smallest of $f(a_i)$ and $f(b_i)$ plus $1/(i+1)$, and so $f(y_i)$ lies between $f(a_i)$ and $f(b_i)$.

[22] $f(a_i) \neq f(b_i)$ because $a_i \neq b_i$.

If $\mathsf{sign}(f(a_i)) \neq \mathsf{sign}(f(b_i))$, then

$$f(y_i) \in (-1, 0) \cup (0, 1)$$

— and so — y_i lies between $f(a_i)$ and $f(b_i)$ which are integers of opposite sign.

Exercise 2.19

Check that also all betweenness constraints (y_i, x, c_i) are satisfied.

This completes the proof that V^* has a total order, namely, since $\mathbb{Q}$ is a total order, the function f defines a total ordering of V^*.

Assume now that there is a total order of V^* — say $\leqslant$. [23] Define a partition $\{A, B\}$ of $V(H)$ by

[23] We're not done yet!

$$A = \{h \mid h \in V(H) \text{ and } h > x\} \text{ and } B = V \setminus A. \quad (2.20)$$

By the betweenness constraints each hyperedge

$$(d_i, e_i) \in E(H)$$

has x between the two endpoints — and so — only one of the two endpoints is in A.

Similarly, there cannot be a hyperedge

$$(a_i, b_i, c_i),$$

with all three elements $< x$ or all three $> x$.

Exercise 2.20

Check all four cases, that no hyperedge $\{a, b, c\} \in E(H)$ is monochromatic.

— We are done — we have shown that given a hypergraph of rank 3 we can construct an instance for the total ordering problem in linear time. The universal set V^* has a total ordering if and only if the hypergraph has a 2–coloring. □

Exercise 2.21

Implement Opatnrý's reduction — Start with the following hypergraph (V, E):

$$V = [n] \tag{2.21}$$

$$E = \{\{i, i+1, i+2\} \mid 1 \leqslant i \leqslant n-2\}. \tag{2.22}$$

Make a list of the betweenness constraints. What is the universal set and what are the total orders that satisfy all the betweenness constraints?

2.5 NP – *Completeness*

Let P be some problem that we wish to solve. — For simplicity — assume that P is a 'yes-or-no' question — or —

a decision problem. [24]

We would like to have a fast algorithm — eg, polynomial — to solve P. The question whether we can do that concerns the — complexity — of the problem.

[24] Usually, the decision variant of a problem is sufficient to solve it. For example, to compute $\omega(G)$ in polynomial time, it would be sufficient to have a polynomial-time algorithm that checks if $\omega(G) \geqslant k$, for each $k \in [n]$.

To acquire some information about the complexity of a decision problem P, it is useful to consider the alternative problem P^*, which is the problem P equipped with an oracle [25] that gives you the answer to P. Then the remaining question that you need to solve is whether the oracle has given you the correct answer.

[25] eg, your teacher

Definition 2.20. The class of problems NP is the class of decision problems P for which an answer — supplied by an oracle — can be tested in polynomial time.

Definition 2.21. A decision problem P is NP – complete if it is in NP and every other problem in NP reduces to it in polynomial time.

Example 2.22. For example, in Section 2.4.2 on Page 35 we reduced the 2 – coloring problem for hypergraphs of rank 3, to the total ordering problem. — Given the fact that — the 2 – coloring problem is NP – complete – we have shown that the total ordering problem is NP – complete as well.

2.5.1 Equivalence covers of splitgraphs

Definition 2.23. A graph is an equivalence graph if it is P_3-free — that is — if it is a disjoint union of cliques. An equivalence cover of a graph G is a set of equivalence subgraphs that covers E(G).

The minimal number of equivalence graphs in a cover of G is denoted as q(G).

Definition 2.24. A graph is a splitgraph if there is a partition of its vertices into a clique and an independent set.

Exercise 2.22

Let G be a splitgraph and let $\{K, S\}$ be a partition of $V(G)$ such that K induces a clique and S induces an independent set. For a vertex $x \in K$ define

$$\delta(x) = |N(x) \cap S| \qquad \text{and let} \qquad D = \max\{\delta(x) \mid x \in K\}.$$

1. Show that $q(G) \geqslant \Delta$: Choose a vertex $x \in K$ that has D neighbors in S. The star induced by x and its neighbors in S has equivalence number D.

2. Show that $q(G) \leqslant D + 1$: To see that enumerate the vertices of S — say $y_1 \cdots y_t$. For each $x \in K$ order its neighbors in S in some arbitrary order. For $i = 1, \cdots, D$, define the equivalence graph with cliques

   $$W_{i,j} = \{y_j\} \cup \{x \in K \mid \text{the } i^{\text{th}} \text{ neighbor of } x \text{ is } y_j\}.$$

   Define one more equivalence graph that consists of one clique; namely K.

In this section we show that computing the equivalence cover number of splitgraphs is NP-complete.

Chromatic index

The chromatic index of a graph is the minimal number that is needed to color the edges of a graph such that no two edges with a nonempty intersection have the same color. The chromatic index is denoted as $\chi'(G)$. By Vizing's theorem the chromatic index of a graph is either Δ or $\Delta + 1$, where Δ is the maximal degree of a vertex in the graph.

Holyer proved the following theorem. [26]

Theorem 2.25. *It is NP-complete to decide whether the chromatic index of a cubic graph is 3 or 4.*

[26] I. Holyer, *The NP-completeness of edge-coloring*, SIAM J. Comput. **10** (1981), pp. 718–720.

A graph is cubic if every vertex has degree 3. For example, the Petersen graph is cubic.

Exercise 2.23

Assume that G is triangle - free. Show that

$$q(G) \quad = \quad \chi'(G).$$

Exercise 2.24

Let G be a cubic graph. Construct a graph G' as follows. Introduce a new vertex x_e for every edge $e \in E(G)$. Make a new vertex x_e adjacent to the two endpoints of e.

Show that

$$\chi'(G) = 3 \quad \Leftrightarrow \quad q(G') = 3.$$

Corollary 2.26. *It is* NP*-complete to decide whether a graph without* K_4 *and with maximal degree at most 6 has equivalence cover number 3 or 4.*

Let G be a cubic graph. Construct a splitgraph G^* as follows.

S1. The splitgraph G^* has a clique $K = V(G)$.

S2. For each $e \in E(G)$ the independent set S of G^* contains two vertices x_e and y_e which are both adjacent to the endpoints of e in K.

S3. For each nonedge $f \in E(\bar{G})$ the independent set S of G^* contains one vertex z_f which is adjacent to the endpoints of f in K.

This completes the description of the splitgraph G^*.

Let's hope it works!

Exercise 2.25

Let G be a cubic graph and let G^* be the graph constructed as above.

$$\chi'(G) = 3 \quad \Leftrightarrow \quad q(G^*) = n + 2$$

where $n = |V(G)|$.

This proves the following theorem.

A. Blokhuis and T. Kloks, *On the equivalence covering number of splitgraphs*, Information Processing Letters **54** (1995), pp. 301–304.

Theorem 2.27. *It is* NP*-complete to decide whether the equivalence cover number of a splitgraph is* D *or* D + 1*. This remains* NP*-complete when the class is restricted so that all vertices in the independent sets of the splitgraphs have degree 2.*

2.6 Lovász Local Lemma

TO SHOW THE EXISTENCE of combinatorial objects the Lovász Local Lemma can be of great use.

TO START consider the 2-coloring problem of a hypergraph $\mathcal{H}$: we wish to color the vertices with two colors such that no hyperedge is monochromatic. Assume that $\mathcal{H}$ is k-uniform — that is — every hyperedge of $\mathcal{H}$ has k vertices.

When $\mathcal{H}$ has less than 2^{k-1} edges then $\mathcal{H}$ is 2-colorable.

To see that, color the vertices of $\mathcal{H}$ independently with probability 1/2 red or black. A BAD EVENT is an hyperedge that is colored monochromatic.

The hypergraph is k-uniform — so — the probability that a bad event occurs is at most 2^{1-k} (either all vertices of the hyperedge are colored black or all vertices are colored red). The probability that some bad event happens is at most their sum and this is less than one (by the assumption on the number of hyperedges). The conclusion is that there is a 2-coloring of $\mathcal{H}$ in which no bad event occurs.

This result is not so great: for a graph it simply says that it is bipartite whenever it has less than two edges (in which case there can be no cycle). It is easy to do better; when each edge intersects at most one other edge then the graph is bipartite also. A generalization of this case is captured by Lovász' local lemma.

Assume that every hyperedge of a k-uniform hypergraph $\mathcal{H}$ intersects at most d other hyperedges. Lovász' local lemma

allows us to conclude that $\mathcal{H}$ is 2-colorable whenever

$$e \cdot (d+1) \quad \leqslant \quad 2^{k-1}$$

(where $e = 2.718\cdots$ is the basis of the natural logarithm).

In a probability space consider a finite set of mutually independent random variables. Let

$$A_1 \quad \cdots \quad A_n$$

be a collection of events. An event is determined by the values of a subset of the variables in the outcome of an experiment. We write $P(A_i)$ for the probability that an event A_i occurs.

The A_i are the (bad) events that we wish to avoid. (In the example above the bad events are monochromatic hyperedges that turn up in the outcome of an experiment which colors the vertices of the hypergraph.)

We can avoid all bad events if we prove

$$P(\cap \bar{A}_i) \quad > \quad 0.$$

When the events are independent then their complements $\bar{A}_i$ are also independent. In that case

$$P(\cap \bar{A}_i) \quad = \quad \prod P(\bar{A}_i) \quad > \quad 0$$

— that is — there exists a way to assign values to the variables such that no bad event happens (unless some A_i surely happens). — On the other hand — it is clearly impossible to avoid all events when some subset of the $\bar{A}_j$s implies some (other) event A_i. Therefore we need some upperbound for the conditional probabilities

$$P(A_i \quad | \quad \bigcap_{j \in J} \bar{A}_j)$$

for any set $J \subseteq [n] \setminus i$.

The local lemma deals with the case where the events are 'almost' independent. To formalize this we make use of a dependency graph. The dependency graph has a vertex for each subset of variables that determines an event. Two vertices are adjacent in the graph when the intersection of the two subsets is nonempty.

Definition 2.28. An event A is independent of a collection of events $B_1 \cdots B_k$ if for all $J \subseteq [k]$ and $J \neq \varnothing$

$$P(A \bigcap_{j \in J} B_j) \quad = \quad P(A) \quad \times \quad P(\bigcap_{j \in J} B_j).$$

Definition 2.29. Let $A_1 \cdots A_n$ be events in a probability space. A graph $D = (V, E)$ with $V = [n]$ is a dependency graph if each event A_i is independent of the collection of events

$$\{ A_j \quad | \quad \{i, j\} \quad \notin \quad E \}.$$

Spencer formulated and proved Lovász' local lemma (originally proved by Lovász and Erdős) as follows.

Lemma 2.30. *Let $A_1 \cdots A_n$ be events with a dependency graph. Let $0 \leqslant x_i < 1$ be real numbers assigned to the events such that*

$$P(A_i) \quad \leqslant \quad x_i \cdot \prod_{\{i,j\} \in E} (1 - x_j).$$

Then

$$\prod P(\cap \bar{A}_i) \quad \geqslant \quad \prod (1 - x_i) \quad > \quad 0.$$

Proof. We first show that for any $J \subseteq [n] \setminus i$

$$P(A_i \mid \cap_{j \in J} \bar{A}_j) \quad \leqslant \quad x_i.$$

This is true when $J = \varnothing$, since

$$P(A_i) \quad \leqslant \quad x_i \cdot \prod_{\{i,j\} \in E} (1 - x_j) \quad \leqslant \quad x_i.$$

We proceed by induction on $|J|$. Let

$$J_1 \quad = \quad \{ j \in J \mid (i, j) \in E. \} \quad \text{and let} \quad J_2 = J \setminus J_1$$

We may assume that $J_1 \neq \varnothing$ otherwise the claim is clearly true. We can write

$$P(A_i \mid \cap_{j \in J} \bar{A}_j) \quad = \quad \frac{P(A_i \cap_{j \in J_1} \bar{A}_j \mid \cap_{j \in J_2} \bar{A}_j)}{P(\cap_{j \in J_1} \bar{A}_j \mid \cap_{j \in J_2} \bar{A}_j)}. \qquad (2.23)$$

The event A_i is independent of the set of events $\{A_j \mid j \in J_2\}$. We use that to find an upperbound for the numerator in (2.23).

$$\begin{aligned} P(A_i \cap_{j\in J_1} \bar{A}_j \mid \cap_{j\in J_2} \bar{A}_j) &\leqslant P(A_i \mid \cap_{j\in J_2} \bar{A}_j) \\ &= P(A_i) \leqslant x_i \cdot \prod_{\{i,j\}\in E} (1-x_j). \end{aligned}$$

To find a lowerbound for the denominator in (2.23); write

$$J_1 = \{j_1, \cdots, j_r\}.$$

Using induction we obtain

$$\begin{aligned} P(\bar{A}_{j_1} \cap \cdots \cap \bar{A}_{j_r} \mid \cap_{j\in J_2} \bar{A}_j) &= \\ P(\bar{A}_{j_1} \mid \cap_{j\in J_2} \bar{A}_j) &\times P(\bar{A}_{j_2} \mid \bar{A}_{j_1} \cap_{j\in J_2} \bar{A}_j) \times \\ \cdots &\times P(\bar{A}_{j_r} \mid \bar{A}_{j_1} \cap \cdots \cap \bar{A}_{j_{r-1}} \cap_{j\in J_2} \bar{A}_j) \\ &\geqslant \prod_{\{i,j\}\in E} (1-x_j). \end{aligned}$$

This proves the claim.

The following observation completes the proof of the lemma.

$$\begin{aligned} P(\cap \bar{A}_i) &= P(\bar{A}_1) \times P(\bar{A}_2 \mid \bar{A}_1) \times \cdots \\ &\times P(\bar{A}_n \mid \bar{A}_1 \cap \cdots \cap \bar{A}_{n-1}) \geqslant \prod_{i=1}^{n} (1-x_i) > 0. \end{aligned}$$

□

Remark 2.31. Assume that all degrees in a dependency graph are at most d. Assume that the probability of any bad event satisfies

$$P(A_i) \leqslant p$$

We claim that if $e \cdot p \cdot (d+1) \leqslant 1$ then $P(\cap \bar{A}_i) > 0$. To see that set $x_i = 1/d+1$. Since the degree of any vertex is at most d, we have

$$\begin{aligned} x_i \cdot \prod_{\{i,j\}\in E} (1-x_j) &\geqslant \frac{1}{d+1} \cdot (1-\frac{1}{d+1})^d \geqslant \\ &\frac{1}{e \cdot (d+1)} \geqslant p. \end{aligned}$$

So the local lemma applies.

Exercise 2.26

Prove the claim we made at the start of this section: Let $\mathcal{H}$ be a k-uniform hypergraph and assume that every hyperedge intersects at most d other hyperedges. When $e(d+1) \leqslant 2^{k-1}$ then $\mathcal{H}$ is 2-colorable.

Remark 2.32. In 2009 Moser and Tardos presented a constructive proof of the Lovász local lemma — that is — they present an algorithm that finds a good object efficiently.

Remark 2.33. Thomassen shows that every hypergraph which is k-regular and k-uniform (that is; every hyperedge has k vertices and every vertex is in k hyperedges) is 2-colorable provided $k \geqslant 4$. The Fano plane shows that not every 3-regular, 3-uniform hypergraph is 2-colorable. The 2-regular graphs that are not 2-colorable are — of course — the odd cycles.

2.6.1 Bounds on dominating sets

The following problem is an example of a problem which can be tackled using the Lovász Local Lemma.

Definition 2.34. Let $a, b \in \mathbb{N}$. A set S of vertices in a graph is an (a, b)-dominating set if every vertex of S is adjacent to at least a vertices in S and every vertex outside S is adjacent to at least b vertices in S.

For a graph G with all degrees at least a let $\gamma_{a,b}(G)$ be the smallest number of elements in an (a, b)-dominating set in G.

Lemma 2.35. *Let* $\frac{1}{2} \leqslant \alpha < 1$. *There exists a number* $R \geqslant 0$ *such that for all* $r > R$ *any* r*-regular graph satisfies*

$$\gamma_{a,b} \quad \leqslant \quad \alpha \cdot n.$$

Proof. Let $N \in \mathbb{N}$ and color (independently) the vertices of an r-regular graph G with N colors. One of the colors appears at

least n/N times among the vertices of G. Say red is such a color and write $V \setminus \mathsf{red}$ for the vertices that are not red.

We CLAIM that

$$V \setminus \mathsf{red}$$

is with positive probability an (a, b)-dominating set.

For a vertex x define a bad event A_x as a coloring of $N[x]$ such that either one of the following holds.

1. x is red and x has less than b neighbors in $V \setminus \mathsf{red}$
2. x is not red and x has less than a neighbors in $V \setminus \mathsf{red}$

The following formula expresses the probability that a bad event A_x occurs.

$$P(A_x) \quad = \quad \frac{1}{N^{r+1}} \cdot \left(\sum_{i=0}^{b-1} \binom{r}{i} (N-1)^i \quad + \right.$$
$$\left. (N-1) \cdot \sum_{i=0}^{a-1} \binom{r}{i} (N-1)^i \right). \quad (2.24)$$

Notice that each event A_x is dependent of at most r^2 other events; namely the events A_y for vertices y at distance at most two from x. — So — by Remark 2.31 (or by Lovász local lemma with all variables $x_i = 1/r^2$) we are done when we show that

$$e \cdot P \cdot r^2 \quad \leqslant \quad 1,$$

where we write $P = P(A_x)$ for the probability of a bad event as in Formula (2.24).

For any given number $N \geqslant 2$ since a and b are fixed numbers in the formula (2.24) the numerator is a polynomial in r and the denominator is an exponential function in r. — So — there exists a number R such that $e \cdot P \cdot r^2 < 1$ whenever $r > R$.

This proves the lemma. □

It is easy to cover the case where G is a graph with minimal degree δ and maximal degree Δ: Use δ to adjust (2.24) and use Δ^2 to bound the degree in the dependency graph.

An $(\mathfrak{a}, \mathfrak{b})$-dominating set that meets the requirements can be constructed efficiently by the algorithm of Moser and Tardos.

2.6.2 The Moser & Tardos algorithm

Moser and Tardos developed (in 2009) a constructive proof of Lovász' local lemma. — To be more precise — let $\mathcal{A}$ be a collection of events with a dependency graph D.

To ease the notations we identify an event A with the set of variables whose outcome determines A.[27]

For an event $A \in \mathcal{A}$ we write $\mathsf{N}(A) \subseteq \mathcal{A} \setminus A$ for its neighbors in the dependency graph D and we write $\bar{\mathsf{N}}(A) = \mathsf{N}(A) \cup \{A\}$ for its closed neighborhood.[28] Let $x : \mathcal{A} \to (0,1)$ be a function which satisfies Lovász' condition

$$\forall_{A \in \mathcal{A}} \quad \mathsf{P}(A) \quad \leqslant \quad x(A) \quad \times \quad \prod_{B \in \mathsf{N}(A)} (1 - x(B)). \qquad (2.25)$$

Moser and Tardos show that a very simple randomized algorithm finds an assignment of the variables which avoids all events $A \in \mathcal{A}$. — Furthermore — the expected number of resampling steps used by this algorithm is bounded by

$$\sum_{A \in \mathcal{A}} \frac{x(A)}{1 - x(A)}.$$

In this section we recapitulate Moser and Tardos' result. In their paper they show furthermore that when the dependency graph has bounded degree and the function x satisfies a slightly stronger condition than (2.25) then there is an 'efficient' deterministic algorithm that finds an assignment of the variables which avoids all events of $\mathcal{A}$. (We refer to their paper for a precise description of this and other results.)

We say that an experiment violates an event when the event happens.

[27] In their paper Moser and Tardos introduce the notation $\mathsf{vbl}(A)$ for this set of variables.

[28] In the dependency graph there is an edge between two events A and B when $A \cap B \neq \varnothing$. So an event A is independent of the collection $\mathcal{A} \setminus (\{A\} \cup \Gamma(A))$. (See Definiton 2.28 on Page 44.)

THE RANDOMIZED ALGORITHM to find an assignment of the variables which avoids all the events of $\mathcal{A}$ is the following.

1. Start with a random asignment of all variables
2. If some event of $\mathcal{A}$ occurs then pick one arbitrarily and find a new random assignment of the variables that it contains
3. Repeat this resampling step until no more bad event occurs.

BELOW we prove the following theorem.

Theorem 2.36. *Let $\mathcal{P}$ be a finite set of mutually independent random variables in a probability space. Let $\mathcal{A}$ be a finite set of events and let x be a function which satisfies the condition of Lovász local lemma (2.25). The algorithm described above resamples an event A an expected number of times at most $x(A)/1-x(A)$ — that is — the expected total number of resampling steps is at most*

$$\sum_{A \in \mathcal{A}} \frac{x(A)}{1-x(A)}.$$

2.6.3 Logs and witness trees

TO PROVE THEOREM 2.36 we show that the algorithm is equivalent to 'checking' sequences of 'witness trees' for the occurrence of bad events.

The execution of the algorithm above produces a log — that is — a sequence of events

$$C : \mathbb{N} \rightarrow \mathcal{A},$$

which are chosen for resampling during the execution. This is a partial function when the algorithm terminates. We assume that there is a fixed (randomized) procedure which selects the bad event for resampling. (This makes the log a random variable.)

Definition 2.37. A witness tree is a finite rooted tree T with a labeling

$$[\,\cdot\,] : \mathsf{V}(\mathsf{T}) \quad \to \quad \mathcal{A}$$

such that for each node $\mathfrak{a}$ its children are labeled by elements of $\bar{\mathsf{N}}([\mathfrak{a}])$.

A witness tree is called proper if all children of a node in a witness tree have different labels.

Given a log C we identify a witness tree with each element $\mathsf{C}(\mathfrak{t})$ as follows. Start with a tree $\mathsf{T}^{\mathfrak{t}}$ which consists of a single root node with label $\mathsf{C}(\mathfrak{t})$. For $\mathfrak{i} = \mathfrak{t}-1, \cdots 1$ distinguish two cases to construct the tree $\mathsf{T}^{\mathfrak{i}}$.

- If there is a vertex $\mathfrak{a}$ in the tree $\mathsf{T}^{\mathfrak{i}+1}$ with $\mathsf{C}(\mathfrak{i}) \in \bar{\mathsf{N}}([\mathfrak{a}])$ then choose $\mathfrak{a}$ such that it is furthest from the root. Attach a new child to $\mathfrak{a}$ and label it as $\mathsf{C}(\mathfrak{i})$.
- If $\bar{\mathsf{N}}(\mathsf{C}(\mathfrak{i})) \cap \mathsf{V}(\mathsf{T}^{\mathfrak{i}+1}) = \varnothing$ then let $\mathsf{T}^{\mathfrak{i}} = \mathsf{T}^{\mathfrak{i}+1}$.

The witness tree $\tau(\mathfrak{t})$ of the resampling step $\mathfrak{t}$ in C is defined as

$$\tau(\mathfrak{t}) \quad = \quad \mathsf{T}^1.$$

Definition 2.38. A witness tree T appears in C if there exists some $\mathfrak{t} \in \mathbb{N}$ such that $\mathsf{T} = \tau(\mathfrak{t})$.

Lemma 2.39. *A witness tree which appears in a log is proper.*

Proof. By definition of the algorithm that produces the witness tree; no two elements of $\mathcal{A}$ that are the same or intersect can appear at the same depth in a witness tree. □

We want to show that the probability that a witness tree T appears in C is at most the probability that it passes a certain test.

Definition 2.40. Let T be a witness tree. A T-check visits the nodes of T in reversed BFS-order, it takes a random evaluation of elements of each node $[\mathfrak{a}]$ in T that it visits and checks if the event $[\mathfrak{a}]$ is violated. The witness tree T passes the check if all events were violated when checked.

Lemma 2.41. *The probability that a witness tree* T *passes its check is*

$$\prod_{\mathfrak{a} \in \mathsf{V}(\mathsf{T})} \mathsf{P}([\mathfrak{a}]).$$

Proof. The random evaluation of the variables in each node is independent of earlier evaluations. □

Lemma 2.42. *Let* T *be a fixed witness tree and let* C *be the random log produced by the algorithm. The probability that* T *appears in the log is at most*

$$\prod_{\mathfrak{a} \in \mathsf{V}(\mathsf{T})} \mathsf{P}([\mathfrak{a}]).$$

Proof. Assume that a random generator produces an infinite sequence of independent random evaluations for each variable P

$$\mathsf{P}^1 \quad \mathsf{P}^2 \quad \dots$$

Whenever the algorithm of Moser and Tardos (or a T-check) calls for a new random sample of P the generator presents the next element in the list.

Assume that a witness tree T appears in the log C — say $\mathsf{T} = \tau(\mathsf{t})$ for some $\mathsf{t} \in \mathbb{N}$. (So the root of T is labeled as $\mathsf{C}(\mathsf{t})$). We need to prove that T passes the T-check.

Let $\mathfrak{a} \in \mathsf{V}(\mathsf{T})$ and let $\mathsf{P} \in [\mathfrak{a}]$. Let $\mathsf{S}(\mathsf{P})$ be the set of nodes $w \in \mathsf{V}(\mathsf{T})$ that are at depth greater than v in T and for which $\mathsf{P} \in [w]$. When the T-check visits v the random evaluation of P produces $\mathsf{P}^{|\mathsf{S}(\mathsf{P})|}$. That is so because the check visits the nodes in order of decreasing depth and no two nodes at the same depth can contain the same element P (since they are disjoint).

Notice that the randomized algorithm of Moser and Tardos resamples the variable P exactly in the same order (by definition of the occurrence of T in the log). Since $[a]$ is violated in the Moser & Tardos algorithm it is also violated in the T-check.

This proves the lemma. □

Let N_A be the number of times that an event $A \in \mathcal{A}$ appears in the log — that is — N_A is the number of times that A is resampled during the execution of the Moser & Tardos algorithm. Then N_A is equal to the number of different witness trees that appear in C and that have their root labeled A. To see that let t_i be such that $C(t_i) = A$ for the i^{th} time. The witness tree $\tau(t_i)$ contains exactly i copies of A — so clearly — $\tau(t_i) \neq \tau(t_j)$ whenever $i \neq j$.

It follows that we can bound the expectation of N_A by summing the bounds on the probabilities of occurrences of witness trees. We do that in the next section.

2.6.4 A Galton - Watson branching process

Wassup ?

In this section we describe and analyze a process that generates proper witness trees with a fixed root $A \in \mathcal{A}$.

Let $x(\cdot)$ be a function satisfying the Lovász' condition 2.25 on Page 48. A process to generate a witness tree is the following.

1. In the first round the process generates a tree with a single node labeled A
2. For each subsequent round choose (independently) an element a of the previous round and (also independently) choose an element $B \in \bar{N}([a])$
3. With probability $x(B)$ add a child node at the node a and give it the label B. With probability $1 - x(B)$: skip.

This process ends when no new vertices are created in some round.

Define

$$x'(B) = x(B) \cdot \prod_{C \in N(B)} (1 - x(C)).$$

Lemma 2.43. *Let* T *be a proper witness tree with its root labeled* A*. The probability that the Galton-Watson process described above produces exactly* T *is*

$$P_T = \frac{1 - x(A)}{x(A)} \cdot \prod_{a \in V(T)} x'([a]).$$

Proof. For $a \in V(T)$ let W_a be the set of elements in $\bar{N}([a])$ that do not appear as the label of a child of a. The probability that the Galton-Watson process produces T is

$$P_T = \frac{1}{x(A)} \cdot \prod_{a \in V(T)} x([a]) \cdot \prod_{u \in W_a} (1 - x([u])).$$

(The leading factor appears because the root is always there.)

We get rid of the W_a's as follows

$$P_T = \frac{1 - x(A)}{x(A)} \cdot \prod_{a \in V(T)} \frac{x([a])}{1 - x([a])} \cdot \prod_{u \in \bar{N}([a])} (1 - x([u])).$$

We can replace inclusive neighborhoods by exclusive ones

$$P_T = \frac{1 - x(A)}{x(A)} \cdot \prod_{a \in V(T)} x([a]) \cdot \prod_{u \in N([a])} (1 - x([u])) = \frac{1 - x(A)}{x(A)} \cdot \prod_{a \in V(T)} x'([a]).$$

This proves the lemma. □

We are now ready to complete the proof of Theorem 2.36 (which is on Page 49).

Proof. Let $\mathcal{T}_A$ be the set of all proper witness trees that have root A. We find for the expected number of times that an event A appears in the log C:

$$\mathbb{E}(N_A) = \sum_{T \in \mathcal{T}_A} P(T \text{ appears in } C) \leqslant \sum_{T \in \mathcal{T}_A} \prod_{a \in V(T)} P([a]) \leqslant \sum_{T \in \mathcal{T}_A} \prod_{a \in V(T)} x'([a]) = \frac{x(A)}{1-x(A)} \cdot \sum_{T \in \mathcal{T}_A} P_T \leqslant \frac{x(A)}{1-x(A)}.$$

The first inequality follows from Lemma 2.42. The second inequality follows from the assumptions in Theorem 2.36. The third inequality follows from Lemma 2.43. The last inequality holds because the Galton-Watson process produces exactly one tree at a time.

This proves the theorem. □

2.7 Szemerédi's Regularity Lemma

The regularity lemma plays an important role in extremal combinatorics. [29]

[29] Extremal graph theory studies maximal or minimal graphs satisfying a certain property.

Let G be a graph and let X and Y be disjoint sets of vertices. Write $e(X,Y)$ for the number of edges that intersect both X and Y. The density of $\{X,Y\}$ is defined as $d(X,Y) = e(X,Y)/|X| \cdot |Y|$.

Definition 2.44. Let X and Y be disjoint sets. The pair $\{X,Y\}$ is ϵ-regular if for all $X' \subseteq X$ and $Y' \subseteq Y$

$$|X'|/|X| \geqslant \epsilon \text{ and } |Y'|/|Y| \geqslant \epsilon \Rightarrow |\, d(X',Y') - d(X,Y) \,| \leqslant \epsilon.$$

Exercise 2.27

Show that $\{X,Y\}$ is ϵ-regular when $d(X,Y) \leqslant \epsilon^3$.

Definition 2.45. A partition $\{V_0, \cdots, V_k\}$ of the vertices of a graph is equitable if

$$|V_i| = |V_j| \quad \text{for all } 1 \leqslant i < j \leqslant k.$$

The set V_0 may be empty; it is called the exceptional class of the partition.

Definition 2.46. An equitable partition $\{V_0, \cdots, V_k\}$ is ϵ-regular if both of the following conditions hold.

(a) $|V_0| \leqslant \epsilon \cdot n$;

(b) all — except at most $\epsilon \cdot k^2$ of the pairs $\{V_i, V_j\}$ $(1 \leqslant i < j \leqslant k)$ —are ϵ-regular.

In this chapter we prove Szemerédi's regularity lemma:

The Regularity Lemma

Lemma 2.47. *Let* $\epsilon \in \mathbb{R}$ *and* $t \in \mathbb{N}$. *There exist* $N, T \in \mathbb{N}$ *such that any graph with at least* N *vertices has an* ϵ*-regular partition* $\{V_0, \cdots, V_k\}$ *with* $t \leqslant k \leqslant T$.

We will assume — throughout this chapter — that

$$0 \quad < \quad \epsilon \quad \leqslant \quad \frac{1}{2}.$$

This is not an important restriction since any ϵ'-regular partition is ϵ-regular for $\epsilon \geqslant \epsilon'$.

Exercise 2.28

Assume G has at most $\epsilon^4 \cdot n^2$ edges. Show that any equitable partition with $|V_0| \leqslant \epsilon \cdot n$ is ϵ-regular.

A partition π' is a refinement of another partition π if every class of π is the union of some classes of π'.

Definition 2.48. Let $\pi = \{V_1, \cdots, V_k\}$ be a partition of the vertices of a graph. The index of π is

$$\mathsf{index}(\pi) \quad = \quad \sum_{1 \leqslant i < j \leqslant k} \frac{|V_i| \cdot |V_j|}{n^2} \cdot d^2(V_i, V_j).$$

Important modification:
When $\pi = \{V_0, \cdots, V_k\}$ is a partition with an exceptional class V_0 then we define $\mathsf{index}(\pi)$ as the index of the refined partition where each element of V_0 forms a class by itself (so $\mathsf{index}(\pi)$ is the index of a partition with $|V_0| + k$ classes).

Exercise 2.29

Show that

$$0 \quad \leqslant \quad \mathsf{index}(\pi) \quad \leqslant \quad \frac{1}{2}.$$

In the search for an ϵ-regular partition we start with an arbitrary equitable partition of the vertices $\{V_0, \cdots, V_t\}$ with

$$|V_0| \quad = \quad t - 1.$$

We refine this partition until it satisfies the conditions. To prove that the number of classes in the final partition is independent of n we use the fact that the index increases whilst it is bounded from above by 1/2.

We first show that the index does not decrease in any refinement. — Below (in Lemma 2.51) — we show that the increase is substantial when there are 'irregular pairs' — that is — pairs $\{V_i, V_j\}$ with subsets V_i' and V_j' that satisfy

$$|V_i'| \quad \geqslant \quad \epsilon \cdot |V_i| \quad \text{and} \quad |V_j'| \quad \geqslant \quad \epsilon \cdot |V_j| \quad \text{and} \quad |d(V_i', V_j') - d(V_i, V_j)| \quad > \quad \epsilon. \quad (2.26)$$

Lemma 2.49. *If π' refines π then*

$$\mathsf{index}(\pi') \quad \geqslant \quad \mathsf{index}(\pi).$$

Proof. Consider a bipartition $\{X, Y\}$. Let $\{X_1, X_2\}$ be a partition of X. Then

$$e(X, Y) \quad = \quad e(X_1, Y) \quad + \quad e(X_2, Y).$$

Rewrite this as

$$|X||Y| \cdot d(X, Y) \quad = \quad |X_1||Y| \cdot d(X_1, Y) \quad + \quad |X_2||Y| \cdot d(X_2, Y).$$

By the Cauchy-Schwartz inequality we obtain

$$d^2(X, Y) \quad \leqslant \quad \frac{|X_1|}{|X|} \cdot d^2(X_1, Y) \quad + \quad \frac{|X_2|}{|X|} \cdot d^2(X_2, Y). \qquad (2.27)$$

Any refinement is obtained by repeated application of bipartitions. This proves the lemma. □

Exercise 2.30

The Cauchy-Schwartz inequality says that for any real numbers a_i and b_i,

$$(a_1 b_1 + \cdots + a_n b_n)^2 \quad \leqslant \quad \left(a_1^2 + \cdots + a_n^2\right) \cdot \left(b_1^2 + \cdots + b_n^2\right).$$

Derive (2.27) (by suitable choice of a_i and b_i).

Lemma 2.50. *Let $\{X, Y\}$ be an irregular bipartition: let $\{X_1, X_2\}$ and $\{Y_1, Y_2\}$ be partitions of X and Y such that*

(a) $|X_1| \geqslant \epsilon \cdot |X|$ *and* $|Y_1| \geqslant \epsilon \cdot |Y|$*;*

(b) $|d(X_1, Y_1) - d(X, Y)| \quad \geqslant \quad \epsilon$*,*

then

$$\sum \frac{|X_i||Y_j|}{|X||Y|} \cdot d^2(X_i, Y_j) \quad \geqslant \quad d^2(X, Y) + \epsilon^4.$$

Proof. Notice that

$$|X||Y| \cdot d(X,Y) \quad = \quad \sum |X_i||Y_j| \cdot d(X_i,Y_j).$$

We have

$$\epsilon^4 \quad \leqslant \quad \sum \frac{|X_i||Y_j|}{|X||Y|} \cdot (d(X_i,Y_j) - d(X,Y))^2 \quad =$$

$$\sum \frac{|X_i||Y_j|}{|X||Y|} \cdot d^2(X_i,Y_j) \quad - \quad 2\,d(X,Y) \sum \frac{|X_i||Y_j|}{|X||Y|} \cdot d(X_i,Y_j) \quad +$$

$$d^2(X,Y) \quad =$$

$$\sum \frac{|X_i||Y_j|}{|X||Y|} \cdot d^2(X_i,Y_j) \quad - \quad d^2(X,Y).$$

This proves the lemma. □

NOTATION: For a partition $\pi = \{V_0, \cdots, V_k\}$ with an exceptional class V_0 let $|\pi| = k$.

Lemma 2.51. *Let $\pi = \{V_0, \cdots, V_k\}$ be an equitable partition with an exceptional class V_0 that satisfies $|V_0| \leqslant \epsilon \cdot n$. Assume that there are more than $\epsilon \cdot k^2$ irregular pairs. There exists a refinement π' of π that satisfies*

$$\mathsf{index}(\pi') \quad > \quad \mathsf{index}(\pi) + \frac{1}{4} \cdot \epsilon^5 \quad \textit{and} \quad |\pi'| \quad \leqslant \quad k \cdot 2^k.$$

Proof. When sets V_i and V_j form an irregular pair $\{V_i, V_j\}$ then there are partitions $\{V_i^1, V_i^2\}$ and $\{V_j^1, V_j^2\}$ such that

$$|V_i^1| \geqslant \epsilon |V_i|, \quad |V_j^1| \geqslant \epsilon |V_j|, \qquad |\,d(V_i^1, V_j^1) - d(V_i, V_j)\,| \quad > \quad \epsilon.$$

By Lemma 2.50 this implies

$$\sum_{k,\ell \in \{1,2\}} \frac{|V_i^k||V_j^\ell|}{n^2} \cdot d^2(V_i^k, V_j^\ell) \quad \geqslant \quad \frac{|V_i||V_j|}{n^2} \cdot d^2(V_i, V_j) + \frac{|V_i||V_j|}{n^2} \cdot \epsilon^4. \tag{2.28}$$

Let π' be the common refinement of these partitions; say this partitions a set V_i as

$$\{\, V_{i,1}, \cdots, V_{i,k_i} \,\}.$$

Then $k_i \leqslant 2^k$ — so — $|\pi'| \leqslant k \cdot 2^k$.

By (2.28) and Lemma 2.49 we have for all irregular pairs $\{V_i, V_j\}$

$$\sum_{a=1}^{k_i} \sum_{b=1}^{k_j} \frac{|V_{i,a}||V_{j,b}|}{n^2} \cdot d^2(V_{i,a}, V_{j,b}) \quad \geqslant \quad \frac{|V_i||V_j|}{n^2} \cdot d^2(V_i, V_j) + \frac{|V_i||V_j|}{n^2} \cdot \epsilon^4.$$

Since there are more than ϵk^2 irregular pairs — and since for $i \geqslant 1$: $|V_i| \geqslant \frac{(1-\epsilon)\cdot n}{k}$ — we find

$$\mathsf{index}(\pi') \quad > \quad \mathsf{index}(\pi) + \epsilon k^2 \cdot \frac{\left((1-\epsilon)n/k\right)^2}{n^2} \cdot \epsilon^4 \quad \geqslant \quad \mathsf{index}(\pi) + \frac{1}{4} \cdot \epsilon^5$$

(since we may assume that $0 < \epsilon \leqslant 1/2$).

This proves the lemma. □

We now prove the regularity lemma (Lemma 2.47 on Page 55).

Proof. Start with an equitable partition $\pi_0 = \{V_0, \cdots, V_t\}$ with $|V_0| = t-1$. We may assume that n is large enough ie $t \leqslant \epsilon \cdot n/2$. [30]

If π_0 is not ϵ-regular there exists a refinement π' which satisfies

$$\mathsf{index}(\pi') \quad \geqslant \quad \mathsf{index}(\pi) + \frac{1}{4} \cdot \epsilon^5 \quad \text{and} \quad |\pi'| \quad \leqslant \quad |\pi| \cdot 2^{|\pi|}.$$

Let $A = |\pi'|$. To make π' into an equitable partition π_1 partition each class further into classes of size exactly $\left\lfloor \frac{1}{4} \cdot \epsilon^6 \cdot \frac{n}{A} \right\rfloor$ and at most one class of size less than that. All the small parts are moved into the exceptional set. This increases the size of the exceptional set by at most $\frac{1}{4} \cdot \epsilon^6 \cdot n$. [31] By Lemmas 2.49 and 2.51:

$$\mathsf{index}(\pi_1) \quad \geqslant \quad \mathsf{index}(\pi_0) + \frac{1}{4} \cdot \epsilon^5.$$

Repeating this process the k^{th} partition π_k satisfies

$$\mathsf{index}(\pi_k) \quad \geqslant \quad \mathsf{index}(\pi_0) + \frac{k}{4} \cdot \epsilon^5.$$

[30] This leaves us some space in the exceptional class — which we need — because the exceptional class grows during the refinements.

[31] That is so because the increase is less than

$$A \cdot \left\lfloor \epsilon^6 \cdot \frac{n}{4A} \right\rfloor \leqslant \frac{1}{4} \cdot \epsilon^6 \cdot n.$$

Notice that for all this to make sense we need $n > 4^A/\epsilon^6$ otherwise the refinement does not exist. The iteration blows up this lower bound on n; but it remains constant.

However, any index is bounded from above by $\frac{1}{2}$. Therefore, the process ends in at most $2 \cdot \epsilon^{-5}$ iterations.

Notice that the increase of the exceptional set is smaller than

$$\frac{1}{4} \cdot \epsilon^6 \cdot n \cdot 2 \cdot \epsilon^{-5} \quad = \quad \frac{1}{2} \cdot \epsilon \cdot n,$$

— that is — since the original exceptional class satisfies

$$|V_0| \quad \leqslant \quad \frac{1}{2} \cdot \epsilon \cdot n,$$

we obtain an ϵ-regular partition. The number of classes is bounded by a function of ϵ and t.

This proves the regularity lemma. □

Let a function $f : \mathbb{N} \to \mathbb{N}$ be defined by $f(1) = t$ and $f(k+1) = \left\lceil \frac{4}{\epsilon^6} \cdot f(k) 2^{f(k)} \right\rceil$. The number of classes in the ϵ-regular partition π_k produced by the algorithm satisfies

$$|\pi_k| \quad \leqslant \quad f(\lceil 2\epsilon^{-5} \rceil).$$

Clearly, this is also the lower bound for n.

Remark 2.52. It is generally not possible to obtain ϵ-regular partitions that do not have any irregular pairs. The following bipartite graph is an example of a graph in which every ϵ-regular partition has irregular pairs. Let

$$A = \{a_1, \cdots, a_n\} \quad \text{and} \quad B = \{b_1, \cdots, b_n\}$$

and let $\{a_i, b_j\} \in E$ if $i \leqslant j$. (This bipartite graph is called a chain graph.)

2.7.1 Construction of regular partitions

Alon et al. showed (in 1994) that it is co-NP-complete to decide whether a given partition of a graph is ϵ-regular. — On the other hand — the lemma can be made constructive; an ϵ-regular partition can be found in $O(M(n))$ time, where $M(n) = n^{2.373}$ is the time needed to multiply two $n \times n$ matrices with entries in $\{0, 1\}$.

The problem to decide whether a given partition is ϵ-regular remains co-NP-complete even when $\epsilon = 1/2$ and $k = 2$. To prove that Alon et al. derive the following lemma (we omit the proof).

Lemma 2.53. *The following problem is* NP*-complete. Given a bipartite graph with color classes* A *and* B *satisfying* $|A| = |B| = n$ *and* $|E| = \frac{1}{2} \cdot n^2 - 1$. *Decide if the graph contains a complete bipartite subgraph* $K_{\frac{n}{2}, \frac{n}{2}}$.

To see that ϵ-regularity is co-NP-complete, Alon et al. make the following observation.

Lemma 2.54. *A bipartite graph with n vertices in each color class and with exactly $\frac{n^2}{2} - 1$ edges contains $K_{\frac{n}{2},\frac{n}{2}}$ if and only if it is not ϵ-regular with $\epsilon = 1/2$.*

Proof. Let the color classes be A and B. Assume $|A| = |B| = n$ and $d(A, B) = \frac{1}{2} - \frac{1}{n^2}$.

Assume the graph contains $K_{\frac{n}{2},\frac{n}{2}}$. Then there are sets X and Y of size $n/2$ and $d(X, Y) = 1$. This implies

$$| d(X, Y) - d(A, B) | > \frac{1}{2},$$

which shows that the graph is not ϵ-regular with $\epsilon = 1/2$.

Assume that the graph is not ϵ-regular with $\epsilon = 1/2$. Then there are sets X and Y of size at least $\frac{n}{2}$ that satisfy

$$| d(X, Y) - \frac{1}{2} + \frac{1}{n^2} | > \frac{1}{2}.$$

Notice that this is only possible when $d(X, Y) = 1$ — that is — $\{X, Y\}$ represents a complete bipartite subgraph (with at least $n/2$ vertices in each color class).

This proves the lemma. □

In their paper Alon et al. prove the following theorem (which is a CONSTRUCTIVE EDITION of the Regularity Lemma (Lemma 2.47 on Page 55).

Theorem 2.55. *For $\epsilon > 0$ and $t \in \mathbb{N}$ there exist $N, T \in \mathbb{N}$ such that every graph with at least N vertices has an ϵ-regular partition with $k + 1$ classes where k satisfies $t \leqslant k \leqslant T$.*

Such a partition can be found in $O(M(n))$ time.

To prove the theorem we need to do some preliminary work.

Definition 2.56. Let A and B be color classes of a bipartite graph satisfying $|A| = |B| = n$. Let d be the average degree. For two vertices $p, q \in B$ define their neighborhood deviation as

$$\sigma(p,q) \quad = \quad |\, N(p) \cap N(q)\,| \quad - \quad \frac{d^2}{n}.$$

For $Y \subseteq B$ $(Y \neq \varnothing)$ define its deviation as [32]

$$\sigma(Y) \quad = \quad \frac{1}{|Y|^2} \cdot \sum_{\substack{p,q \in Y \\ p \neq q}} \sigma(p,q).$$

A constructive version of the regularity lemma depends on the construction of suitable subsets V_i' and V_j' satisfying (2.26) (of irregular pairs $\{V_i, V_j\}$). These 'witnesses' to the irregularity are hard to find.

The following lemma shows that it is possible to 'approximate' witnesses.

Lemma 2.57. *Let a bipartite graph have color classes A and B which satisfy $|A| = |B| = n$. Let d be the average degree. Let $0 < \epsilon < 1/16$. Assume there exists a set $Y \subseteq B$ such that*

$$|Y| \quad \geqslant \quad \epsilon \cdot n \quad \textit{and} \quad \sigma(Y) \quad \geqslant \quad \frac{\epsilon^3}{2} \cdot n \qquad (2.29)$$

Then one of the following items holds true

I. $d < \epsilon^3 \cdot n$;

II. $|\,\{y \mid y \in B \quad \textit{and} \quad |\deg(y) - d\,| \quad \geqslant \quad \epsilon^4 \cdot n\}\,| \quad > \quad \frac{\epsilon^4}{8} \cdot n$;

III. *There exist subsets $A' \subseteq A$ and $B' \subseteq B$ that satisfy*

$$|A'| \quad \geqslant \quad \frac{\epsilon^4}{4} \cdot |A| \quad \textit{and} \quad |B'| \quad \geqslant \quad \frac{\epsilon^4}{4} \cdot |B| \quad \textit{and}$$
$$|\, d(A',B') \,-\, d(A,B)\,| \quad > \quad \epsilon^4.$$

[32] When the graph is complete bipartite or empty then

$$\sigma(p,q) = 0$$

for all pairs $p, q \in B$. In that case $\sigma(Y) = 0$ for all nonempty sets $Y \subseteq B$.

As another example, define a bipartite graph H on the 7 points and 7 lines of the Fano plane. A point and line are adjacent of they are incident in the plane. We have an average degree $d = 3$ since any point lies on 3 lines. Since any pair of points lie on one line

$$\sigma(p,q) = 1 - 9/7 = -\frac{2}{7}.$$

When B is the set of points of the Fano plane, we find

$$\sigma(B) = \frac{1}{49} \cdot 7 \cdot 6 \cdot (-\frac{2}{7}) = -\frac{12}{49}.$$

There exists an $O(M(n))$*-algorithm that produces sets* A' *and* B' *as specified in* III *when* I *and* II *do not hold.*

Proof. Assume that I and II do not happen. We prove III.

Define

$$Y' \quad = \quad \{\, y \in Y \quad | \quad |\deg(y) - d| \quad < \quad \epsilon^4 \cdot n \,\}.$$

Then $Y' \neq \varnothing$ since II does not occur.

Choose $y_0 \in Y$ that maximizes $\sum_{y \in Y} \sigma(y, y_0)$. By taking an average we estimate $\sum_{y \in Y} \sigma(y, y_0)$.

$$\sum_{y' \in Y'} \sum_{\substack{y \in Y \\ y \neq y'}} \sigma(y, y') \;=\; \sigma(Y) \cdot |Y|^2 \quad - \quad \sum_{y' \in Y \setminus Y'} \sum_{\substack{y \in Y \\ y \neq y'}} \sigma(y, y')$$

$$\geqslant \quad \frac{\epsilon^3}{2} \cdot n \cdot |Y|^2 \quad - \quad \frac{\epsilon^4}{8} \cdot n \cdot |Y| \cdot n.$$

(Here we use Y's property that $\sigma(Y) \geqslant \epsilon^3 \cdot n/2$ and the assumption that II does not occur which implies $|Y \setminus Y'| \leqslant \frac{\epsilon^4}{8} \cdot n$.)
Since $|Y'| \leqslant |Y|$ and $|Y| \geqslant \epsilon \cdot n$, we have

$$\sum_{y \in Y} \sigma(y, y_0) \quad \geqslant \quad \frac{\epsilon^3}{2} \cdot n \cdot |Y| - \frac{\epsilon^4}{8} \cdot n^2 \quad \geqslant \quad \frac{3}{8} \cdot \epsilon^3 \cdot n \cdot |Y|. \quad (2.30)$$

We CLAIM

$$|\,\{y \quad | \quad y \in Y \quad \text{and} \quad \sigma(y, y_0) \quad > \quad 2\epsilon^4 \cdot n\}\,| \quad \geqslant \quad \frac{\epsilon^4}{4} \cdot n. \quad (2.31)$$

OTHERWISE

$$\sum_{y \in Y} \sigma(y, y_0) \quad < \quad \frac{\epsilon^4}{4} \cdot n^2 + |Y| \cdot 2\epsilon^4 \cdot n$$

$$\leqslant \quad \frac{\epsilon^3}{4} \cdot n \cdot |Y| + 2\epsilon^4 \cdot n \cdot |Y| \quad \leqslant \quad \frac{3}{8} \cdot \epsilon^3 \cdot n \cdot |Y|$$

and this contradicts (2.30).

It follows from (2.31) that there exists a set $B' \subseteq Y \setminus \{y_0\}$ which satisfies

$$|B'| \geqslant \frac{\epsilon^4}{4} \cdot n \quad \text{and} \quad \forall_{b \in B'} \quad |\,N(b) \cap N(y_0)\,| \quad > \quad \frac{d^2}{n} + 2\epsilon^4 \cdot n.$$

Let $A' = N(y_0)$. Since $d \geqslant \epsilon^3 \cdot n$ and $16\epsilon < 1$,

$$|A'| \quad \geqslant \quad d - \epsilon^4 \cdot n \quad \geqslant \quad \epsilon^3 \cdot n - \epsilon^4 \cdot n \quad \geqslant \quad 15\epsilon^4 \cdot n \quad \geqslant \quad \frac{\epsilon^4}{4} \cdot n.$$

We CLAIM

$$|\, d(A', B') - d(A, B)\,| \quad > \quad \epsilon^4.$$

TO SEE THAT notice

$$e(A', B') = \sum_{b \in B'} |\, N(y_0) \cap N(b)\,| \quad > \quad \left(\frac{d^2}{n} + 2\epsilon^4 \cdot n \right) \cdot |B'|.$$

It follows that

$$\begin{aligned} d(A', B') - d(A, B) \quad &> \quad \left(\frac{d^2}{n} \quad + \quad 2\epsilon^4 \cdot n \right) \cdot \frac{1}{|A'|} \quad - \quad \frac{d}{n} \\ &> \quad \frac{d^2}{n(d + \epsilon^4 \cdot n)} + 2\epsilon^4 - \frac{d}{n} \quad = \\ &\qquad 2\epsilon^4 \quad - \quad \frac{d\epsilon^4}{d + \epsilon^4 \cdot n} \quad \geqslant \quad \epsilon^4. \end{aligned}$$

(In the second line we use the fact that $|A'| \leqslant n$ since $A' \subseteq A$ and the fact that $|A'| = |N(y_0)| < d + \epsilon^4 \cdot n$ since $y_0 \in Y'$.)

AN ALGORITHM to compute the sets A' and B' proceeds by computing for all $y_0 \in B$ with $|\, \deg(y_0) - d\,| < \epsilon^4 \cdot n$ the set

$$B(y_0) \quad = \quad \{y \quad | \quad y \in B \quad \text{and} \quad \sigma(y, y_0) \quad > \quad 2\epsilon^4 \cdot n\}.$$

There exists a vertex y_0 with $|B(y_0)| \geqslant \frac{\epsilon^4}{4} \cdot n$ (see (2.31)). The required sets are $B' = B(y_0)$ and $A' = N(y_0)$.
All quantities $\sigma(y, y')$ can be obtained by squaring the adjacency matrix.

This proves the lemma. □

Exercise 2.31

Let d be the average degree of a bipartite graph with color classes A and B of size n. Assume $d < \epsilon^3 \cdot n$ (Case I of Lemma 2.57). Prove that the graph is ϵ-regular.

HINT: See Exercise 2.27.

Exercise 2.32

Let the color classes A and B of a bipartite graph satisfy $|A| = n$ and $|B| = n$. Show that the graph is ϵ-regular if for all $X \subseteq A$ and $Y \subseteq B$

$$|X| = \lceil \epsilon \cdot |A| \rceil \quad \text{and} \quad |Y| = \lceil \epsilon \cdot |B| \rceil \quad \Rightarrow \quad |\, d(X,Y) - d(A,B)\,| \quad \leqslant \quad \epsilon.$$

HINT: Compare Definition 2.44 (which is on Page 54).

Lemma 2.58. *Let A and B be color classes of a bipartite graph and assume that $|A| = |B| = n$. Assume*

$$2n^{-\frac{1}{4}} \quad < \quad \epsilon \quad < \quad \frac{1}{16}.$$

Assume that case II*, in Lemma 2.57, does not occur, ie*

$$|\{y \in B \mid |\deg(y) - d| \quad \geqslant \quad \epsilon^4 \cdot n\}| \quad \leqslant \quad \frac{\epsilon^4}{8} \cdot n.$$

If the graph is <u>not</u> ϵ-regular then there exists a set $Y \subset B$ which satisfies (2.29) ie

$$|Y| \quad \geqslant \quad \epsilon \cdot n \quad \textit{and} \quad \sigma(Y) \quad \geqslant \quad \frac{\epsilon^3}{2} \cdot n. \tag{2.32}$$

Proof. Assume that no set $Y \subseteq B$, $|Y| \geqslant \epsilon \cdot n$ satisfies (2.32). We show that the graph is ϵ-regular.

Let $X \subseteq A$ and $Y \subseteq B$ and let

$$|X| = \lceil \epsilon \cdot |A| \rceil \quad \text{and} \quad |Y| = \lceil \epsilon \cdot |B| \rceil .$$

We show that

$$|\, d(X,Y) - d(A,B)\,| \quad \leqslant \quad \epsilon.$$

We CLAIM

$$\sum_{x \in X} \left(|\, N(x) \cap Y\,| - \frac{d}{n} \cdot |Y| \right)^2 \quad \leqslant \quad e(A,Y) + |Y|^2 \cdot \sigma(Y) + \frac{2}{5}\epsilon^5 \cdot n^3.$$

TO SEE THAT write $(m_{i,j})$ for the bipartite adjacency matrix. Then

$$\sum_{x\in X}\left(|N(x)\cap Y|-\frac{d}{n}\cdot|Y|\right)^2 \quad\leqslant$$

$$\sum_{x\in A}\left(|N(x)\cap Y|-\frac{d}{n}\cdot|Y|\right)^2 \quad =$$

$$\sum_{x\in A}\left(\sum_{y\in Y}m_{x,y}-\frac{d}{n}\cdot|Y|\right)^2 \quad = \quad \sum_{x\in A}\left(\sum_{y\in Y}m_{x,y}^2+\frac{d^2}{n^2}\cdot|Y|^2+\right.$$

$$\left.\sum_{\substack{y,y'\in Y\\ y\neq y'}}m_{x,y}\cdot m_{x,y'}-\frac{2d}{n}\cdot|Y|\cdot\sum_{y\in Y}m_{x,y}\right) \quad =$$

$$e(A,Y)+\frac{d^2}{n}\cdot|Y|^2+\sum_{\substack{y,y'\in Y\\ y\neq y'}}|\,N(y)\cap N(y')\,|-\frac{2d}{n}\cdot|Y|\cdot e(A,Y) \quad =$$

$$e(A,Y)+\frac{d^2}{n}\cdot|Y|^2+\sum_{\substack{y,y'\in Y\\ y\neq y'}}\left(\sigma(y,y')+\frac{d^2}{n}\right)-\frac{2d}{n}\cdot|Y|\cdot e(A,Y) \quad\leqslant$$

$$e(A,Y)+\sigma(Y)\cdot|Y|^2+\frac{2d^2}{n}\cdot|Y|^2-\frac{2d}{n}\cdot|Y|\cdot e(A,Y).$$

To prove the claim, we must show

$$\frac{d^2}{n}\cdot|Y|^2-\frac{d}{n}\cdot|Y|\cdot e(A,Y) \quad\leqslant\quad \frac{\epsilon^5}{5}\cdot n^3,$$

that is,

we need to show that: $\boxed{d(A,Y) \quad\geqslant\quad \frac{d}{n}-\frac{\epsilon^5\cdot n^3}{5d}\cdot\frac{1}{|Y|^2}.}$

To see that notice

$$d(A,Y)=\frac{e(A,Y)}{n\cdot|Y|}\geqslant\frac{(d-\epsilon^4\cdot n)(|Y|-\frac{\epsilon^4}{8}\cdot n)}{n\cdot|Y|} \quad =$$

$$\frac{d}{n}-\epsilon^4-\frac{\epsilon^4\cdot d}{8\cdot|Y|}+\frac{\epsilon^8\cdot n}{8\cdot|Y|} \quad\geqslant\quad \frac{d}{n}-\epsilon^4-\frac{\epsilon^3}{8}.$$

(We used $|Y|\geqslant\ \epsilon\cdot n\geqslant\epsilon\cdot d$.)

We now use the assumption that $\epsilon^4\cdot n>1$ and that $|Y|\leqslant\epsilon\cdot n+1$:

$$\frac{\epsilon^5\cdot n^3}{5d\cdot|Y|^2} \quad\geqslant\quad \frac{\epsilon^5\cdot n^2}{5(\epsilon\cdot n+1)^2} \quad\geqslant\quad \frac{\epsilon^5}{5(\epsilon+\epsilon^4)^2} \quad\geqslant\quad \frac{\epsilon^3}{8}+\epsilon^4.$$

This proves the claim.

By the Cauchy-Schwartz inequality we have

$$\sum_{x\in X}\left(|\,N(x)\cap Y\,|-\frac{d}{n}\cdot|Y|\right)^2 \quad\geqslant\quad \frac{1}{|X|}\left((\sum_{x\in X}|\,N(x)\cap Y\,|)-\frac{d}{n}\cdot|X|\cdot|Y|\right)^2.$$

— So — by the previous claim

$$\left((\sum_{x\in X}|\,N(x)\cap Y\,|)-\frac{d}{n}\cdot|X|\cdot|Y|\right)^2 \quad\leqslant\quad |X|\cdot\left(e(A,Y)+|Y|^2\cdot\sigma(Y)+\frac{2\epsilon^5}{5}\cdot n^3\right).$$

When we divide by $|X|^2\cdot|Y|^2$ we obtain

$$|\,d(X,Y)-d(A,B)\,| \quad\leqslant\quad \frac{1}{|X|\cdot|Y|}\cdot\left(e(A,Y)+|Y|^2\cdot\sigma(Y)+\frac{2\epsilon^5}{5}\cdot n^3\right).$$

We now use that

(i) $e(A,Y)\leqslant(d+\epsilon^4\cdot n)\cdot|Y|+\frac{\epsilon^4}{8}\cdot n^2$

(ii) $\sigma(Y)\leqslant\frac{\epsilon^3}{2}\cdot n$

(iii) $\epsilon>2n^{-1/4}$,

and we find

$$\begin{aligned}&|\,d(X,Y)-d(A,B)\,|^2 \quad\leqslant\\ &\frac{1}{|X|\cdot|Y|^2}\cdot\left((d+\epsilon^4\cdot n)\cdot|Y|+\frac{\epsilon^4\cdot n^2}{8}+\frac{\epsilon^3\cdot n}{2}\cdot|Y|^2+\frac{2\epsilon^5}{5}\cdot n^3\right) \quad\leqslant\\ &\frac{n+\epsilon^4\cdot n}{\epsilon^2\cdot n^2}+\frac{\epsilon^4\cdot n^2}{8\epsilon^3\cdot n^3}+\frac{\epsilon^3\cdot n}{2\epsilon\cdot n}+\frac{2\epsilon^5\cdot n^3}{5\epsilon^3\cdot n^3} \quad\leqslant\\ &\frac{\epsilon^2\cdot(1+\epsilon^4)}{16}+\frac{9\epsilon^2}{10}+\frac{\epsilon^5}{128} \quad\leqslant\quad \epsilon^2.\end{aligned}$$

This proves that the graph is ϵ-regular — that is — it proves the lemma. □

Corollary 2.59. *Let a bipartite graph* H *have color classes* A *and* B *that satisfy*

$$|A| \quad = \quad |B| \quad = \quad n.$$

Assume

$$2 \cdot n^{-1/4} \quad < \quad \epsilon \quad < \quad \frac{1}{16}.$$

There exists an algorithm that runs in $O(M(n))$ *time and that verifies that* H *is* ϵ*-regular or finds sets* $A' \subseteq A$ *and* $B' \subseteq B$ *that satisfy*

$$|A'| \geqslant \frac{\epsilon^4}{16} \cdot n, \quad |B'| \geqslant \frac{\epsilon^4}{16} \cdot n, \quad |\, d(A', B') - d(A, B)\,| \geqslant \epsilon^4. \quad (2.33)$$

Proof. If $d \leqslant \epsilon^3 \cdot n$, then we are done; we can check that in $O(n^2)$ time and report that H is ϵ-regular.

Next if

$$|\{\, y \mid y \in B \quad |\deg(y) - d| \geqslant \epsilon^4 \cdot n\,\}| \quad \geqslant \quad \frac{\epsilon^4}{8} \cdot n,$$

then the degrees of at least half of them deviate from d in the same direction. — That is — we can find a set $B' \subseteq B$ that satisfies $|B'| \geqslant \frac{\epsilon^4}{16} \cdot n$ and that satisfies

$$|\, d(A, B') - d(A, B)\,| \quad \geqslant \quad \epsilon^4.$$

In that case we are done.

Finally, we need to show that we can find A' and B' as required when H is not ϵ-regular — that is (by Lemma 2.58) —when there exists a set Y that satisfies Equation 2.29 ie

$$|Y| \quad \geqslant \quad \epsilon \cdot n \quad \text{and} \quad \sigma(Y) \quad \geqslant \quad \frac{\epsilon^3}{2} \cdot n. \qquad (2.34)$$

By Lemma 2.57 there exists an $O(M(n))$ algorithm to compute these sets A' and B' when Y exists.

This proves the claim. □

Alon et al proceed to formulate the key-Lemma 2.51 (on Page 58) in a constructive form.

Lemma 2.60. *Let* $k \in \mathbb{N}$ *and let* $0 < \gamma \leqslant 1/2$. *Let* $\Pi_0 = \{C_0, \cdots, C_k\}$ *be an equitable partition and assume*

$$|C_1| \quad > \quad 4^{2k} \quad \text{and} \quad |C_0| \quad \leqslant \quad \gamma \cdot n. \tag{2.35}$$

Assume that proofs are given as part of the input of more than $\gamma \cdot k^2$ *pairs of classes showing that they are* <u>*not*</u> γ*-regular.* [33] *There is an algorithm that runs in* $O(n)$ *time that produces a refinement* Π_1 *of* Π_0 *satisfying*

$$|\Pi_1| \quad = \quad k \cdot 4^k.$$

Furthermore, the exceptional class increases by at most $n/2^k$ *and*

$$\mathsf{index}(\Pi_1) \quad > \quad \mathsf{index}(\Pi_0) + \frac{1}{4} \cdot \gamma^5.$$

[33] By 'proofs' we mean 'witnesses' ie subsets of the color classes that violate γ-regularity.

Proof. By Lemma 2.51 there is a linear-time algorithm that computes a refinement Π' of Π_0 (leaving the exceptional class unaltered but which may have other classes of unequal size) satisfying

$$|\Pi'| \quad \leqslant \quad k \cdot 2^k \quad \text{and} \quad \mathsf{index}(\Pi') \quad > \quad \mathsf{index}(\Pi_0) + \frac{1}{4} \cdot \gamma^5.$$

Since $|C_1| > 4^{2k}$ we can refine Π' into an equitable partition Π_1 which satisfies [34] (by Lemma 2.49)

$$|\Pi_1| \quad = \quad k \cdot 4^k \quad \text{and} \quad \mathsf{index}(\Pi_1) \quad > \quad \mathsf{index}(\Pi_0) + \frac{1}{4} \cdot \gamma^5.$$

This increases the size of the exceptional class of Π_0 (and of Π') by at most $n/2^k$. □

[34] To obtain Π_1 partition the parts of Π' further into parts of size

$$\lfloor (n - |C_0|)/k \cdot 4^k \rfloor = \lfloor |C_1|/4^k \rfloor .$$

This increases the exceptional class by at most

$$|\Pi'| \cdot \left\lfloor \frac{n - |C_0|}{k \cdot 4^k} \right\rfloor \quad \leqslant \quad \frac{n}{2^k}.$$

Below follows the proof of Theorem 2.55:

Theorem. *For* $\epsilon > 0$ *and* $t \in \mathbb{N}$ *there exist* $N, T \in \mathbb{N}$ *such that every graph with at least* N *vertices has an* ϵ*-regular partition with* $k+1$ *classes where* k *satisfies* $t \leqslant k \leqslant T$.

Such a partition can be found in $O(M(n))$ *time.*

Proof. Let $\epsilon > 0$ and $t \in \mathbb{N}$.

The following algorithm computes a sequence of equitable partitions — say (Π_i). Write $|\Pi_i| = k_i$ and let n_i be the size of a class of Π_i that is not exceptional. Let $\gamma = \epsilon^4/16$.

1. Start with an equitable partition $\Pi_1 = \{C_0, \cdots, C_{k_1}\}$ with

$$|C_0| < k_1 \quad \text{and} \quad |C_i| = n_1 = \lfloor n/k_1 \rfloor \quad \text{for } i > 0.$$

(Below we determine a suitable value for k_1.) Set $i := 1$.

2. To proceed from a partition Π_i use Corollary 2.59 to determine for every pair of classes in P_i whether they are ϵ-regular or else to find pairs of subsets (of size at least $\frac{\epsilon^4}{16} \cdot n_i$) as in Equation 2.33. [35]

3. If at most $\epsilon \cdot k_i$ pairs are not ϵ-regular then report Π_i as an ϵ-regular partition. STOP.

4. Otherwise call on Lemma 2.60 to compute a refinement Π_{i+1} of Π_i satisfying

$$k_{i+1} = k_i \cdot 4^{k_i} \quad \text{and} \quad \mathsf{index}(\Pi_{i+1}) \geqslant \mathsf{index}(\Pi_i) + \frac{1}{4} \cdot \gamma^5.$$

Write C_0^i for the exceptional class of Π_i. [36] Then the exceptional class of Π_{i+1} satisfies

$$|C_0^{i+1}| - |C_0^i| \leqslant \frac{n}{2^{k_i}}.$$

Notice that to apply Lemma 2.60 we need to ensure Equation 2.35 — ie —

$$n_i > 4^{2k_i} \quad \text{and} \quad |C_0^i| \leqslant \gamma \cdot n.$$

5. Set $i := i + 1$ and go to Step 2.

NOTICE THAT

$$\mathsf{index}(\Pi_i) \geqslant \mathsf{index}(\Pi_1) + (i-1) \cdot \frac{\gamma^5}{4}.$$

So — since the index cannot exceed $1/2$ — the algorithm ends in at most $1 + \lfloor 2 \cdot \gamma^{-5} \rfloor$ iterations.

[35] To apply the corollary we need to ensure that n is large enough to satisfy the condition $\epsilon^4 \cdot n_i > 16$ ie $\gamma \cdot n_i > 1$.

[36] To compute Π_{i+1} use Lemma 2.51 to compute Π' (on input Π_i) with

$$|\Pi'| \leqslant k_i \cdot 2^k_i$$

Refine Π' into parts of size

$$n_{i+1} = \left\lfloor \frac{n - |C_0^i|}{k_i \cdot 4^{k_i}} \right\rfloor.$$

This increases the exceptional class

$$|C_0^{i+1}| - |C_0^i| \leqslant k_i \cdot 2^{k_i} \cdot \left\lfloor \frac{n - |C_0^i|}{k_i \cdot 4^{k_i}} \right\rfloor \leqslant \frac{n}{2^{k_i}}.$$

To define a lowerbound for the number of vertices in the graph introduce a function $f : \mathbb{N} \to \mathbb{N}$ as follows

$$f(i) \quad = \quad \begin{cases} k_1 & \text{if } i = 1 \\ f(i-1) \cdot 4^{f(i-1)} & \text{otherwise.} \end{cases}$$

Let $T = f\left(1 + \lfloor 2 \cdot \gamma^{-5} \rfloor\right)$, let $N = \max\{ 2{\cdot}T{\cdot}4^{2\cdot T}, \frac{2\cdot T}{\gamma} \}$ and assume that $n \geqslant N$.

We show that the exceptional class does not exceed $\gamma \cdot n$. We may assume that

$$\gamma \cdot 2^{k_1} \quad > \quad 4. \tag{2.36}$$

Claim: $|C_0^i| \leqslant \gamma \cdot n \cdot (1 - 1/2^i)$. For $i = 1$ we have

$$n \quad \geqslant \quad N \quad \geqslant \quad \frac{2 \cdot T}{\gamma} \quad \geqslant \quad \frac{2 \cdot k_1}{\gamma} \quad \Rightarrow$$
$$|C_0^1| \quad < \quad k_1 \quad \leqslant \quad \frac{\gamma \cdot n}{2}.$$

This show that the claim is true for $i = 1$.

We have

$$|C_0^{i+1}| - |C_0^i| \quad \leqslant \quad \frac{n}{2^{k_i}} \quad \text{and we show:} \quad \frac{n}{2^{k_i}} \quad < \quad \gamma \cdot \frac{n}{2^{i+1}}.$$

This is true for $i = 1$ by the assumption (2.36) and

$$k_{i+1} = k_i \cdot 4^{k_i} \quad \Rightarrow \quad \gamma \cdot 2^{k_{i+1}} \quad > \quad 2 \cdot \gamma \cdot 2^{k_i} \quad \geqslant \quad \gamma \cdot 2^{i+1}.$$

This proves the claim.

Corollary 2.59 requires that

$$\gamma \cdot n_i \quad > \quad 1.$$

To see that this holds true observe

$$n_i \quad \geqslant \quad \frac{(1-\gamma) \cdot n}{k_i} \quad \geqslant \quad \frac{(1-\gamma) \cdot N}{T} \quad \geqslant \quad 2 \cdot \frac{1-\gamma}{\gamma}.$$

Since $\gamma < 1/2$ this implies

$$\gamma \cdot n_i \quad \geqslant \quad 2 \cdot (1-\gamma) \quad > \quad 1.$$

Lemma 2.60 requires that

$$n_i \quad > \quad 4^{2k_i}.$$

We have

$$n_i \geqslant {}^{(1-\gamma)\cdot N}/_{k_i} \quad \text{and} \quad k_i \leqslant T \quad \text{and} \quad N \geqslant 2 \cdot T \cdot 4^{2 \cdot T}.$$

It easily follows that

$$\frac{n_i}{4^{2 \cdot k_i}} \quad \geqslant \quad 2 \cdot (1-\gamma) \quad > \quad 1.$$

This proves the theorem. □

Remark 2.61. In 2014 J. Fox and L. M. Lovász proved tight lowerbounds for the number of parts in an ϵ-regular partition.

2.8 Edge - thickness and stickiness

For a set S let

$$\Lambda(S) \quad = \quad \{\lambda : S \to \mathbb{R}^{\geqslant 0} \mid \sum_{x \in S} \lambda(x) \, = \, 1\}.$$

The elements of $\Lambda(S)$ are called thickness functions. For a thickness function λ and $A \subseteq S$ write $\lambda(A) = \sum_{x \in A} \lambda(x)$.

Definition 2.62. Let G be a graph. The thickness of G is defined as

$$\phi(G) \quad = \quad \inf_{\lambda \in \Lambda(V)} \quad \sup_{e \in E} \quad \lambda(e).$$

T. Kloks, C. Lee and J. Liu, *Stickiness, edge-thickness, and clique-thickness in graphs*, Journal of Information Science and Engineering **20** (2004), pp. 207–217.

The inf and sup may be replaced by min and max.

Exercise 2.33

Assume that G is a graph which is not empty. Show that

$$\frac{\delta}{m} \quad \leqslant \quad \phi(G) \quad \leqslant \quad \frac{1}{\alpha(G)},$$

where m is the number of edges and δ is the minimal degree of a vertex in G and $\alpha(G)$ is the size of a largest independent set in G.

Exercise 2.34

(i) CLEARLY when G has an isolated vertex then its thickness is 0.

(ii) Assume that G has no isolated vertices. Let λ be a thickness function that realizes the thickness of G. Denote $\phi = \phi(G)$. Show that every vertex is incident with an edge of weight ϕ.

(iii) Assume that G has no isolated vertices. Show that there exists a thickness function λ that realizes the thickness of G and that has a range

$$\{0, \phi, \frac{\phi}{2}\}.$$

Definition 2.63. The stickiness of a graph G is

$$s(G) = \max_{X \subseteq V} |X| - |\bar{N}(X)|$$

where $\bar{N}(X) = \cup_{x \in X} N(x)$.

A graph is Hallian if $s(G) \leqslant 0$. A graph is Hallian if and only if there is a partition of V into a matching and a set of odd cycles.

Exercise 2.35

When $s(G) > 0$ then it is realized by an independent set.

Exercise 2.36

If G has an isolated vertex then $\phi(G) = 0$ otherwise

$$\phi(G) = \frac{2}{n + s(G)}.$$

Computing stickiness

Let G be a graph. Construct a bipartite graph G' as follows. Create two copies of a vertex $x \in V$ — say x^1 and x^2. For $A \subseteq V$ write $A^i = \{x^i \mid x \in A\}$. The set of vertices of G' is $V^1 \cup V^2$. The edges of G' are

$$E(G') = \{\{x^1, y^2\} \mid \{x, y\} \in E(G)\}.$$

Construct a flow - network on G' by giving each edge of G' a capacity ∞. Add one source - vertex s and make it adjacent to all vertices of V^1. All edges that are incident with s have capacity 1. Add one sink - vertex t and make it adjacent to all vertices of V^2. All edges incident with t have capacity 1.

A cut in the network is a set of the form $A^1 \cup B^2 \cup \{s\}$ where $A \subseteq V$ and $\bar{N}(A) \subseteq B$. The capacity of such a cut is

$$|V \setminus A| + |B| = n + |B| - |A|.$$

A cut of minimum capacity satisfies $B = \bar{N}(A)$ — and so — it minimizes $|\bar{N}(A)| - |A|$.

A minimum cut can be computed $O(n \cdot m)$ time.

J. Orlin, *Max flows in* $O(nm)$ *time, or better*, Proceedings STOC'13, pp. 765–774.

Remark 2.64 (Motzkin - Straus Theorem (1965)). Let G be a graph with vertex set $[n]$. Give each vertex i a values $x_i \geqslant 0$ such that $\sum_i x_i = 1$. The largest value over these assignments of

$$\sum_{(i,j)\in E} x_i \cdot x_j$$

is equal to $\frac{1}{2} \cdot (1 - 1/\omega)$.

2.9 Clique Separators

Definition 2.65. Let G be a graph. A set $S \subseteq V$ is a clique separator if $S = \varnothing$ or a clique and $G - S$ is disconnected.

Whitesides showed that one can find a clique cutset in $O(n^3)$ time. In this section we present a variation of this algorithm. We show that there exists an $O(n^4)$ algorithm that lists all minimal clique separators.

Definition 2.66. Let G be a graph. A set $S \subseteq V$ is a minimal clique separator if S is a minimal separator — and — either $S = \varnothing$ or a clique.

Example 2.67. Notice that the number of clique separators that a graph may have is exponential. — For example — consider the graph built up from a clique K_t and a path P_3, say $[x, y, z]$. Add additional edges from the midpoint y to all vertices of the clique K_t. Then, for any subset $W \subseteq V(K_t)$

$$W \cup \{y\} \quad \text{is a clique separator.}$$

This shows that the graph has at least 2^t clique separators, and $t+3$ vertices.

On the other hand the graph has only one minimal separator — namely — $\{y\}$.

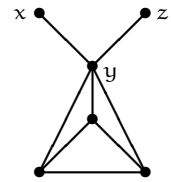


Figure 2.6: This graph has an exponential number of clique separators, as the size of the clique goes to ∞.

For a graph G, write $\sigma(G)$ for the number of minimal clique separators in G.

Lemma 2.68. *For any graph* $\sigma < |V|$.

Proof. We claim that, when G is connected, $\sigma(G) < n$. Notice that this implies the lemma. — In order to prove this — we may assume that G is connected. We may also assume that G is not a clique — as otherwise $\sigma(G) = 0 < n$.

Let $x \in V$ be a vertex for which the largest component in $G - N[x]$ is as large as possible. Let C be this largest component. Let

$$S = N(C) \quad \text{and} \quad X = V \setminus N[C]. \tag{2.37}$$

Then $x \in X$ — so $X \neq \varnothing$. Since G is connected $S \neq \varnothing$, and since G is not a clique $C \neq \varnothing$ — so $\{X, S, C\}$ is a partition of the set V.

— By our choice of x — every vertex of X is adjacent to every vertex of S. This implies that every minimal clique separator in $G[X \cup S]$ either contains X or else contains S. — So —

Here, we let a set stand in for the graph induced by that set.

$$\sigma(X \cup S) = \begin{cases} \sigma(X) & \text{if } S \text{ is a clique and } X \text{ is not} \\ \sigma(S) & \text{if } X \text{ is a clique and } S \text{ is not} \\ 0 & \text{if neither or both are cliques.} \end{cases} \tag{2.38}$$

Any vertex of S has a neighbor in C, and since C is connected, no two vertices in S are separated by a clique separator. This reduces the number (2.38) of minimal clique separators in $X \cup S$ that are clique separators in G, to at most $\sigma(X)$.

A minimal clique separator which is contained in $C \cup S$ may no longer be a minimal separator in $G - X$. — To repair that — in the graph $C \cup S$, add all edges between pairs in S.[37] Denote this graph as $C \cup \bar{S}$.

Some minimal clique separators in $C \cup \bar{S}$ may not be cliques in G[38] — but — as an upperbound for the number of minimal clique separators — that are contained in $C \cup S$ — we find $\sigma(C \cup \bar{S})$.

Using induction on the number of vertices in the graph we find:

$$\begin{aligned} \sigma(G) &\leqslant \sigma(X) + \sigma(C \cup \bar{S}) \\ &< |X| + |C \cup S| \qquad (\text{since } C \cup S \text{ is connected}) \\ &= n. \end{aligned}$$

This proves the lemma. □

[37] A clique separator T, contained in $C \cup S$, has all vertices of S that are not in T in one component of $G - T$. Removal of the set X may disconnect this component and possibly T is not a minimal separator in $G - X$.

[38] In our algorithm, we will sift the minimal clique separators of $C \cup \bar{S}$, and select those that are cliques in G.

Exercise 2.37

Prove the first inequality. Hint: Show that every minimal clique separator is one in $G[X \cup S]$ or one in $G[C \cup S]$.

2.9.1 Feasible Partitions

To turn the proof of Lemma 2.68 into an algorithm we need to find the partition $\{X, S, C\}$ as mentioned there. — That is — we need a feasible partition of $V(G)$ — as defined below.

Definition 2.69. A partition $\{X, S, C\}$ of V is feasible if

(a) $G[C]$ is connected and

(b) $S = N(C)$ and S separates X and C and

(c) every vertex of X is adjacent to every vertex of S.

In this section we show how to find a feasible partition. The idea is to start with $C = \{R\}$, for some arbitrary vertex $R \in V$ which is not universal.[39]

[39] A vertex is universal if it it adjacent to all others.

As an invariant for the algorithm we let $\{X, S, C\}$ be a partitition of V with the property

$$C \text{ is connected and } S = N(C) \text{ and } S \text{ separates } X \text{ and } C. \tag{2.39}$$

Notice that the partition $\{X, S, C\}$ satisfies (2.39) at the initialization — when $C = \{R\}$ — since R is nonuniversal.[40]

[40] If every vertex is universal, then G is a clique, and then, there is no minimal separator.

To make progress — towards a feasible partition — let C grow as follows:

$$\text{Choose } y \in S \text{ with } X \not\subset N(y). \tag{2.40}$$

The vertex y is added to C.

Notice that, when such a vertex $y \in S$ does not exist, we may conclude the postcondition — that is — $\{X, S, C\}$ is a feasible partition.

The weakest precondition is a property that ensures progress when valid, and that ensures the postcondition when not valid. In our case, the weakest precondition is the existence of a vertex y as in (2.40).

This proves the correctness of Algorithm 5.

Exercise 2.38

Let $T(n)$ denote the worst-case running-time bound of algorithm 5. Prove that

$$T(n) \leqslant T(n-1) + O(n^2) \quad \Rightarrow \quad T(n) = O(n^3). \tag{2.41}$$

```
1: procedure FP(G)
2:
3:     if G is a clique then
4:         there is no partition
5:     else
6:         R ←∈ { x | x ∈ V  and  d(x) < n − 1 }
7:         C ← {R}
8:         S ← N(R)
9:         X ← V \ N[C]
10:
11:        while ∃_{y∈S} X ⊄ N(y) do
12:            C ← C ∪ {y}
13:            S ← ( S \ {y} ) ∪ ( N(y) ∩ X )
14:            X ← X \ N(y)
15:        end while
16:
17:        report {X, S, C}
18:    end if
19:
20: end procedure
```

Algorithm 5: Feasible Partition

2.9.2 Intermezzo

Another way to describe the inclusion of the vertex y in C is by contractions.

Definition 2.70. Let $\{x, y\} \in E(G)$. The contraction of $\{x, y\}$ is the operation that replaces the two vertices x and y by one new vertex — say xy — whose neighborhood is defined as

$$N(xy) = (N(x) \cup N(y)) \setminus \{x, y\}.$$

The operation that includes y in C could be replaced by contracting the edge $\{R, y\}$.

Back to Business:

Theorem 2.71. *There exists an algorithm — that runs in $O(n^4)$ time — and that computes all minimal clique separators in a graph G.*

Proof. The algorithm we propose computes a feasible partition $\{X, S, C\}$ of $V(G)$. When S is a clique, it recursively computes $\sigma(X)$. All minimal clique separators in $G[X]$, with S tagged on, are minimal clique separators in G.

Next, all edges are added to make S a clique. Denote the subgraph induced by $C \cup S$ as $C \cup \bar{S}$. Recursively, count the minimal clique separators in $C \cup \bar{S}$ that are cliques in G.

The recursions take place on subsets of V that form a partition of $V(G)$, namely X and $C \cup S$. The overhead is dominated by the computation of the feasible partition, which, by (2.41), takes $O(n^3)$ time.

This proves that the time used by this algorithm is $O(n^4)$.

This completes the proof. □

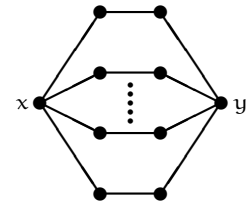


Figure 2.7: This example shows that the number of minimal separators in a graph can be exponential.

Remark 2.72. The number of minimal separators in a graph may be exponential. — For example — consider the graph in Figure 2.7 on the previous page. An $x\,|\,y$–separator must contain a vertex of each $x\sim y$–path. If there are t such paths, then there are 2^t minimal $x|y$–separators and the graph has $2t+2$ vertices.

There exists an algorithm that lists all minimal separators in a graph with polynomial delay. That is, the algorithm produces all minimal separators, and it spends polynomial time — either to produce a next one or to end.

Exercise 2.39

Let G be a graph. Consider the following set R of betweenness relations :

$$(x, y, z) \in R \quad \Leftrightarrow$$
$$\text{there is a minimal clique separator that}$$
$$\text{contains } y \text{ and separates } x \text{ and } z. \qquad (2.42)$$

Design an algorithm that finds a total ordering $\leqslant$ of V satisfying

$$(a, b, c) \in R \quad \Leftrightarrow \quad a < b < c \quad \text{or} \quad c < b < a.$$

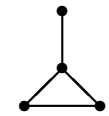


Figure 2.8: The figure shows the paw. It has no induced P_4 and no induced C_4. It does have a universal vertex.

2.9.3 Another Intermezzo: Trivially perfect graphs

When a graph G is connected but has no induced C_4 and no induced P_4 then it has a UNIVERSAL VERTEX (that is a vertex adjacent to all others). This was proved by Wolk in 1961. Let G be a connected graph without induced C_4 or P_4. Then there exists a rooted tree T with $V(T) = V(G)$ such that any two vertices x and y are adjacent in G if and only if one lies on the path to the root of the other one.

The graphs without induced P_4 nor C_4 were given the epithet 'trivially perfect' by Golumbic. The reason for that name is that in any induced subgraph of the graph the independence number equals the number of maximal cliques.

2.10 Vertex ranking

Definition 2.73. Let G be a graph and let $t \in \mathbb{N}$. A t-ranking is a coloring $c : V \to [t]$ which satisfies the following property. For any two vertices x and y with $c(x) = c(y)$ any $x \sim y$-path contains a vertex z with $c(z) > c(x)$.

The rank of G is denoted as $\chi_r(G)$ and it is the smallest t for which G has a t-ranking. — Clearly — $\chi \leqslant \chi_r$ since a t-ranking is a proper coloring.

Exercise 2.40

Let G be a connected graph and assume it has a t-ranking. There exists at most one vertex that has color t.

Exercise 2.41

What is the rank of C_4? What is the rank of P_4? What is $\chi_r(G)$ when G is trivially perfect?

Exercise 2.42

Let G be a graph which is not a clique. Show that

$$\chi_r(G) \quad = \quad \min_S \max_C |S| + \chi_r(C)$$

where S varies over the set of minimal separators in G and C varies over the collection of components of $G - S$.

2.10.1 Permutation graphs

Definition 2.74. A graph is a permutation graph if it is the intersection graph of a set of line-segments in the plane that have their endpoints on two parallel lines.

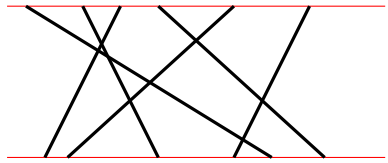


Figure 2.9: The figure shows a permutation model. The graph has the black line-segments as vertices; two being adjacent if the line-segments intersect.

Exercise 2.43

(1) Show that a graph is a permutation graph if and only if its complement is a permutation graph.

(2) Show that a permutation graph is a comparability graph — that is it has a transitive orientation of its edges. [41]

(3) Show that a graph is a permutation graph if and only if it and its complement are comparability graphs.

(4) Design efficient algorithms to compute α and ω for permutation graphs. You may assume that a permutation model is given as input.

(5) Show that any permutation graph has a dominating pair — that is — a pair of vertices x and y with the property that any $x \sim y$-path is a dominating set in the graph. [42]

[41] That is, if $x \to y$ and $y \to z$ then $\{x, z\} \in E$ and $x \to z$.

[42] A set $D \subseteq V$ is a dominating set if every vertex that is not in D has a neighbor in D.

2.10.2 Separators in permutation graphs

Consider a scanline in a permutation model — that is — a new line-segment s with its endpoints on the two parallel lines. The line-segment s splits the set of vertices in three parts:

(a) vertices of which the line-segment lies to the left of s

(b) vertices of which the line-segment lies to the right of s

(c) vertices of which the line-segment crosses with s.

Exercise 2.44

Let G be a permutation graph and let S be a minimal separator in G. In any model for G there exists a scanline s such that the vertices of S are the line-segments that cross s.

Corollary: the number of minimal separators in a permutation graph is $O(n^2)$.

Hint: Consider a permutation model. Remove the vertices of S from the model. Let C_1 and C_2 be two components of $G - S$ such that every vertex of S has a neighbor in both. For the scanline s take any line-segment in the model that lies between any two line-segments of C_1 and C_2 and that crosses no other line-segments. That line-segment must exist since C_1 and C_2 form connected parts in the diagram. Since a vertex of S has a neighbor in C_1 and in C_2 it must cross s. The only vertices that were removed from the model are in S so all line-segments that cross s are in S.

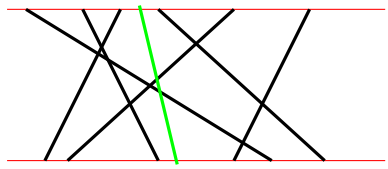


Figure 2.10: Illustration of a green scanline.

2.10.3 Vertex ranking of permutation graphs

Definition 2.75. Let s_1 and s_2 be two parallel scanlines in a permutation model. The piece $C(s_1, s_2)$ is the subgraph of G induced by the following sets of vertices.

i. vertices of which the line-segment s between s_1 and s_2

ii. vertices that cross at least one of s_1 and s_2.

Definition 2.76. Let $C(s_1, s_2)$ be a piece. A scanline t splits the piece if t is between s_1 and s_2.

Theorem 2.77. *There exists an* $O(n^6)$ *algorithm to compute the rank of a permutation graph.*

Proof. A permutation model can be constructed in linear time via a modular decomposition. [43]

Organize the pieces according to the number of vertices that are in it. To compute χ_r for a piece $C(s_1, s_2)$ try all scanlines that split the piece into smaller pieces. By Exercise 2.42 the rank of the piece can be computed from the maximal rank of a smaller piece and the size of the separator. □

[43] R. McConnell and J. Spinrad, *Modular decomposition and transitive orientation*, Discrete Mathematics **201** (1999), pp. 189–241.

2.11 Cographs

Definition 2.78. A cograph is a graph without induced P_4.

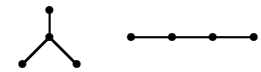


Figure 2.11: The figure shows the claw, ie, $K_{1,3}$, on the left and P_4 on the right.

One important observation is that cographs are closed under complementation. — That is — when G is a cograph then so is its complement $\bar{G}$. That is so because

$$P_4 \text{ is isomorphic to its complement } \bar{P}_4.$$

— One other thing — a graph is a cograph if and only if all of its components induce cographs. That is so because an induced P_4 cannot have points in more than one component. Thus, whenever G_1 and G_2 are two cographs, we can create two new cographs by taking their union $G_1 \oplus G_2$ and their join $G_1 \otimes G_2$. [44] Folklore asserts that this property characterizes cographs.

Theorem 2.79. *A graph is a cograph if and only if every induced subgraph has only one vertex — or else — it or its complement is disconnected.*

[44] In the union $G_1 \oplus G_2$ the edges are just the edges of G_1 and G_2. In the join $G_1 \otimes G_2$ we add all edges that have one endpoint in G_1 and the other in G_2. So

$$\bar{G}_1 \otimes \bar{G}_2 = \overline{G_1 \oplus G_2}.$$

By Theorem 2.79 each cograph G is represented by a cotree — that is — a rooted tree T and a bijection from the leaves of T to $V(G)$. The internal nodes of T are labeled: each internal node has a label $\oplus$ or $\otimes$. Two vertices in G are adjacent if and only if their lowest common ancestor in T is labeled with $\otimes$.

Definition 2.80. A pair of vertices x and y in a graph is a twin if every third vertex z is either adjacent to both or neither of x and y — that is — a twin is a module in the graph with two elements. A twin is a true twin if the pair is adjacent and a false twin if the pair is not adjacent.

Theorem 2.81. *A graph is a cograph if and only if every induced subgraph with at least two vertices has a twin.*

Exercise 2.45

Characterize bipartite cographs.

Exercise 2.46

Design an efficient algorithm to compute the rank χ_r of a cograph.

2.11.1 Switching cographs

Definition 2.82. A two-graph is a pair (V, Δ) where V is a finite set and where Δ is a collection of triples in V with the property that every set with 4 elements from V contains an even number of triples that are in Δ. [45]

[45] Two-graphs look a lot like graphs but in two-graphs the 'edges' are triples.

Exercise 2.47

Let G be a graph and let Δ be the 'odd triples in G' — that is — the set of triples in V that carry an odd number of edges between them. Show that this is a two-graph.

Exercise 2.48

Let G be a graph and let $S \subseteq V$. A switch of G with respect to S replaces all edges that have one end in S by nonedges and all nonadjacent pairs with one end in S by edges. Show that a switch does not change the set of odd triples.

Call two graphs G and H switch-equivalent if one is obtained from the other via a switch. Van Lint and Seidel showed that two-graphs are switch equivalence - classes.

Exercise 2.49

A two-graph is the switch equivalence - class of a cograph if and only if it does not contain the pentagon — that is — the set of odd triples of the C_5.

The switch - class of cographs is characterized by the following property. There exists a tree T without vertices of degree two. Let V be the set of leaves in T. Since T is bipartite it has a unique coloring with two colors — say black and white. Define Δ as the collection of triples $\{x, y, z\}$ in V if the paths that connect the three meet in a black vertex. Then (V, Δ) is a two-graph and the two-graphs obtained in this manner are exactly the switching - class of cographs.

Exercise 2.50

Show that a graph can be switched to a cograph if and only if it does not contain C_5, the bull, the gem or the cogem as an induced subgraph — that is — no subgraph with 5 vertices switches to C_5.

Show that a graph G can be switched to a cograph if and only if every induced subgraph with at least two vertices has a twin or an anti-twin. [46]

[46] An anti-twin in a graph is a pair of vertices x and y such that every third vertex z is adjacent to exactly one of them.

Exercise 2.51

Design a linear - time algorithm to compute χ for graphs in the switch - class of cographs. You may assume that a tree - model is given as input.

HINT: Show that $\chi = \omega$ for any graph that switches to a cograph.

Exercise 2.52

The rank of the adjacency matrix of a cograph is equal to the number of distinct non - zero rows. [47]
For a proof of this see eg the following paper.

[47] For graphs in general this is an upperbound.

G. Royle, The rank of a cograph, *Electronic Journal on Combinatorics* **10**, (2003), Note 11.

In this paper the question is raised whether any other "natural" classes of graphs satisfy this property. What can you say about the rank of a graph that switches to a cograph?

HINT: P_5 has eigenvalue zero — so — the rank of P_5 is not equal to the number of distinct rows in the adjacency matrix of P_5. — Furthermore — P_5 switches to a cograph.

Switching does not change the eigenvalues of the Seidel - matrix of the graph. The Seidel - matrix has:

1. zeros on the diagonal
2. -1 at entries that represent edges in the graph
3. $+1$ at entries that represent nonedges in the graph.

So it is the matrix $\mathsf{J} - \mathsf{I} - 2\mathsf{A}$ where A is the ordinary adjacency matrix of the graph. When the graph is regular the eigenspaces of A and $\mathsf{J} - \mathsf{I} - 2\mathsf{A}$ are the same.

Seidel's favorite graph $\mathsf{C}_5 + \mathsf{K}_1$ has Seidel spectrum $\{\sqrt{5}^{(3)}, -\sqrt{5}^{(3)}\}$. This graph switches to the net (the complement of the 3-sun).

The spectrum of the Seidel matrix is the same for any two graphs that are switching - equivalent. For a proof see Corollary 3.3 in the paper below.

J. Seidel, *A survey of two - graphs*, Atti Convegno Internazionale Teorie Combinatorie, Tomo 1 (Rome, Italy, September 3-15, 1973), Acdemia Nazionale dei Lincei, Roma (1976), pp. 481–511.

2.12 Parameterized Algorithms

LET'S START WITH A BIRD'S - EYE VIEW.

Exercise 2.53

Show that there is an algorithm that checks if a graph G has a set $S \subseteq V$ — $|S| \leqslant k$ — such that $G - S$ is a cograph — where $k \in \mathbb{N}$ is a parameter. Your algorithm should run in

$$O\left(4^k \cdot n^3\right).$$

The algorithm is a function of k.

Hint: First design an $O\left(n^3\right)$ algorithm that finds an induced P_4 — if there exists one — eg, using a matrix multiplication. Next, if P is an induced P_4, then branch, each time putting a different point of P in S. Since $|S| \leqslant k$, the depth of the recursion is k.

Unless $P = NP$ no NP - complete problem can be solved in polynomial time. All known algorithms that solve NP-complete problems are exponential. — Therefore — we wish to design 'fast' exponential - time algorithms to solve hard problems.

I've heard that before!

Parameterized algorithmics is a theory designed to help you do this. The genesis of the theory is the notion of a parameterized problem.

Definition 2.83. A parameterized problem is a pair (P, k) where P is a computational problem and k is a parameter that differentiates some solutions from others.

The parameter expresses the 'size' of a solution but the definition of a size is up to the composer of the problem. — Below — we give three examples of parameterized problems.

When the values of the parameter k ranges over $\mathbb{N}$ then a parameterized problem is an ordering of the solutions (by their size).

1. Let I be the vertex cover - problem: Let G be a graph. Find a smallest set $S \subset V(G)$ such that $G - S$ is empty.

   Let $k \in \mathbb{N}$. The parameterized vertex cover - problem (I, k) is to find a vertex cover S with $|S| \leqslant k$.

$G - S$ is empty if

$$E(G - S) = \varnothing$$

2. Let II be the edge domination - problem: Let G be a graph. An edge dominating set is a dominating set in the linegraph $L(G)$ — in other words — it is a set $M \subseteq E(G)$ of edges in G such that $E(G - V(M)) = \varnothing$. The problem asks for a smallest edge dominating set.
   The parameterized problem edge domination - problem (II, k) is to find an edge dominating set with at most k edges.

   For a set M of edges let $V(M)$ be the set of endpoints of edges in M — that is — $V(M) = \cup_{e \in M} e$.

3. Problem III is the feedback vertex set - problem. This problem applies to graphs G that may have loops and multiple edges. The problem is to find a smallest set $S \subset V(G)$ such that $G - S$ has no cycles.
   The parameterized problem parameterized feedback vertex set problem (III, k) is to find a feedback vertex set of size at most k.

Definition 2.84. A fixed parameter - algorithm is an algorithm that solves a parameterized problem (P, k) with parameter $k \in \mathbb{N}$ in

$$O\left(f(k) \cdot |P|^c \right)$$

time. Here $f : \mathbb{N} \to \mathbb{N}$ is a computable function and $c \in \mathbb{N}$.

For ease of discussion in Definition 2.84 we assume that the parameter k ranges over the natural numbers.

A computable function is a function which can be evaluated via an algorithm.

A constant is a natural number.

Notice that c is a constant; (not a parameter); the run - time of a fixed parameter algorithm is a polynomial in the size of the instance of problem P. The influence of the parameter k on the run - time is some arbitrary function $f(k)$. The algorithm runs in polynomial - time if we let the parameter k be a constant (then $f(k)$ disappears in the Big-Oh).

When P is NP - complete the function f is not a polynomial.

There are problems that can not be solved by a fixed parameter - algorithm. The 'good guys' are called fixed parameter - tractable.

Definition 2.85. Let (P, k) be a parameterized problem. The problem is fixed parameter tractable (FPT) if there is a fixed parameter - algorithm that solves (P, k).

Nobody knows of an algorithm that solves the parameterized problem $\omega \geqslant k$ and that runs in time $O(n^c)$ where c does not depend on k. The clique problem is $W[1]$ - hard.

Remark 2.86. In their book Downey and Fellows introduce a W-hierarchy to capture the hardness of parameterized problems. In this hierarchy $W[0] = \mathsf{FPT}$. A parameterized problems is $W[1]$ - hard if there is no fixed - parameter algorithm to solve it (under certain logical assumptions).

This section has too many imprecise definitions and — unexplained — assumptions. Let's hope that — in practice — it all works out fine.

2.13 The bounded search technique

Let's take a look at a basic technique to design fixed parameter - algorithms. The bounded search technique is best explained by example.

2.13.1 Vertex cover

Our example is the vertex cover problem parameterized by the size of the solution. Let G be a graph and let S be a vertex cover in G — that is

$$e \in E(G) \quad \Rightarrow \quad e \cap S \neq \varnothing.$$

We want a vertex cover with at most k vertices.

The following algorithm searches for a vertex cover of size $\leqslant k$ (where the parameter k ranges over $\mathbb{N} \cup \{0\}$).

Every edge of the graph has at least one endpoint that is in a vertex cover. The search tree tries both endpoints and searches for a small vertex cover that contains an endpoint.

If the graph has no edges then $\varnothing$ is a solution. If $k = 0$ and $E(G) \neq \varnothing$ then there is no solution. Otherwise pick an edge e from the graph and build a search tree with that edge as a root.

The search tree (with root e) branches at the root into two subtrees; each subtree is labeled with an endpoint of e. The subtrees are evaluated as follows.

The selected endpoint of e is put in S and this endpoint is deleted from the graph. The parameter k decreases with 1 and the subtree searches for a vertex cover of size $k-1$ (in the remaining graph).

The size of the search tree is $O(2^k)$ (since the depth of the search tree is at most k and every node has at most two children). This proves the following theorem.

Theorem 2.87. *The vertex cover problem is fixed-parameter tractable and can be solved in* $2^k \cdot |I|^{O(1)}$ *time.*

2.13.2 Edge dominating set

The parameterized vertex cover problem is easy to solve via the building of a search tree of size 2^k.

A DIFFERENT KETTLE OF FISH is the parameterized edge dominating set problem. To build a search tree for this problem we would want to find a set of edges M which satisfies the following two conditions.

1. $|M|$ is bounded by some function of k
2. any solution to the parameterized edge domination problem has at least one edge in M.

This road doesn't look very appealing. We take a different approach.

Why doesn't this idea work? What happens to the problem if we assume that the degree of the graph is at most 3?

Instead of trying to locate the edges of a solution we first find a collection of suitable sets $S \subseteq V(G)$ that are endpoints of a solution. Step two is to find a a solution — that is — a set of edges that solves the parameterized edge domination problem and that contains all elements of S.

Exercise 2.54

Let M be a minimum edge dominating set. Then $V(M)$ is a vertex cover.

Exercise 2.55

Prove or disprove:
Let S be a minimal vertex cover. A minimum edge dominating set M which satisfies $S \subseteq V(M)$ can be computed in polynomial time

as follows. Initialize M as a maximum matching in $G[S]$. For each vertex $x \in S \setminus V(M)$ add one edge of G that contains x to M.

The exercises show that our job is done if we can find the right minimal vertex cover.

Theorem 2.88. *The parameterized edge dominating set problem is fixed-parameter tractable and can be solved in* $4^k \cdot |II|^{O(1)}$ *time.*

Proof. The following algorithm solves the parameterized edge domination problem.

1. generate the set $\mathcal{S}$ of all minimal vertex covers with at most $2k$ elements
2. for each $S \in \mathcal{S}$ compute a minimum edge dominating set M with $S \subseteq V(M)$ as in Exercise 2.55
3. Output M when $|M| \leqslant k$.

This proves the theorem. □

2.13.3 Feedback vertex set

In the previous two examples we made use of the fact that a small 'local' part of the graph contain a solution. The feedback vertex problem lacks this property.

A solution is some 'small' set of vertices that hits all cycles of the graph. What happens to the feedback vertex set problem if we assume that the graph has no induced cycles of length more than 3?

IT IS ALWAYS A GOOD IDEA to reduce the graph before going on a venture that takes exponential time — so — that's what we do.

1. if a vertex is not in any cycle then we delete the vertex from the graph
2. if there are two vertices with more than two edges running between them then we delete one of those edges

Exercise: Show that the reductions take polynomial time and that they are 'safe:' the graph has a solution if and only if the reduced graph has a solution.

3. if a vertex is in a loop then we delete the vertex from the graph and we decrease the parameter k by 1 (the vertex is in every solution)

4. if x has only one neighbor y and if there are at least two edges between x and y then we remove x and we add a loop at y

5. if a vertex is in exactly two edges that connect it to two different neighbors then we remove the vertex and replace it with an edge that connects the two neighbors.

Definition 2.89. A graph is reduced if none of the operations above apply.

A reduced graph has no loops and at most two parallel edges between any two vertices. Furthermore every vertex is in at least three edges.

We turn our attention to the reduced graph. A solution is some 'small' set X of vertices that hits all cycles. If we remove X from the graph the remainder is a 'large' forest F. All the vertices of F that have at most two neighbors in F must have neighbors in X. Not every vertex of X has a small degree; that is so because there is no large forest with only a few leaves and only a few vertices of degree two.

Let's work this out. Order the vertices in the graph in a descending order of their degree say

$$d(v_1) \geqslant \cdots \geqslant d(v_n).$$

Define the set of high - degree vertices as the set of the first $\lceil 3k/2 \rceil$ vertices in this order:

$$H = \{v_1, \ldots, v_{\lceil 3k/2 \rceil}\}.$$

Call the elements of H the vertices of high degree.

Lemma 2.90. *Let* $G = (V, E)$ *be a reduced graph. Any feedback vertex set in* G *of size at most* k *contains at least one vertex of high degree.*

Proof. Let X be a feedback vertex set and let $F = V \setminus X$. Then $G[F]$ is a forest and the number of edges in $G[F]$ is at most $|F| - 1$. We have that

$$\sum_{z \in X} d(z) \quad \geqslant \quad |E| - |F| + 1.$$

Assume that there is a feedback vertex set X of size at most k and assume that $X \cap H = \varnothing$. We show that this leads to a contradiction.

Following the idea outlined above we concentrate on the number of edges that run between F and X.

Let $f = |F \setminus H|$. Each vertex in $F \setminus H$ has degree in G at least three since G is reduced. Let $a = d(v_{\lceil 3k/2 \rceil})$. Then $a \geqslant 3$.

By assumption $H \subseteq F$. We have that

$$\sum_{z \in F} d(z) \quad \geqslant \quad \lceil 3k/2 \rceil \cdot a \; + \; 3 \cdot f.$$

The induced graph $G[F]$ is a forest and the number of edges in $G[F]$ is at most $|F| - 1$. Thus, the number of edges that run between F and X is at least

$$\lceil 3k/2 \rceil a + 3f - |F| + 1 \quad = \quad \lceil 3k/2 \rceil (a - 1) + 2f + 1.$$

On the other hand each vertex in X has degree at most a — and so — the number of edges between X and F is at most $k \cdot a$.

So we have that

$$\lceil 3k/2 \rceil (a - 1) + 2f + 1 \quad \leqslant \quad k \cdot a.$$

and this leads to $a < 3$. This contradicts the fact that each vertex in G has degree at least 3. □

Lemma 2.90 allows the following parameterized algorithm for feedback vertex set.

Reduce the graph. The vertices of high degree in the reduced graph form the root of a branching. A selected vertex is put in the solution set and deleted from the graph.

Each branching operation results in at most $\lceil 3k/2 \rceil$ subproblems. The depth of the recursion is at most k. This proves the following theorem.

Exercise: Check all this.

Theorem 2.91. *The parameterized feedback vertex set problem is fixed-parameter tractable and can be solved in* $(1.5k)^k \cdot |III|^{O(1)}$ *time.*

Exercise 2.56

A set of vertices in a graph is called a P_3 - cover if each path of length two in the graph contains at least one vertex from the set.

Please design a fixed parameter algorithm for the problem to decide if a graph has a P_3 - cover of size at most k (where k is a parameter).

Hint: Any P_3 - cover contains at least one of the three vertices of any path of two edges.

2.13.4 Further reading

The fastest parameterized algorithm for the vertex cover problem runs in $1.2738^k n^{O(1)}$ time. This was obtained by Chen, Kanj and Xia in 2010.

The edge dominating set problem can be solved in $2.3147^k n^{O(1)}$ time by Xiao, Kloks and Poon. This is further improved to $2.2351^k n^{O(1)}$ by Iwaide and Nagamochi.

For the feedback vertex set problem, there is a $2.7^k n^{O(1)}$-time randomized algorithm by Li and Nederlof. There is also a $3.46^k n^{O(1)}$-time deterministic algorithm by Iwata and Kobayashi.

References

R. Downey and M. Fellows, *Fixed Parameter Tractability and Completeness*. Dagstuhl workshop Complexity theory: current research, 1992, pp. 191 – 225.

J. Chen, I. Kanj and G. Xia, *Improved upper bounds for vertex cover*, Theor. Comput. Sci. 411 (2010), pp. 3736 – 3756.

M. Xiao, T. Kloks and S. Poon, *New parameterized algorithms for the edge dominating set problem*, Theor. Comput. Sci. 511 (2013), pp. 147 – 158.

K. Iwaide and H. Nagamochi, *An Improved Algorithm for Parameterized Edge Dominating Set Problem*, J. Graph Algorithms Appl. 20 (2016), pp. 23 – 58.

J. Li and J. Nederlof, *Detecting Feedback Vertex Sets of Size* k *in* $O*(2.7^k)$ *Time*, Proceedings of SODA 2020, pp. 971 – 989.

Y. Iwata and Y. Kobayashi, *Improved Analysis of Highest-Degree Branching for Feedback Vertex Set*, Algorithmica 83(8) (2021), pp. 2503 – 2520.

2.14 Matchings

A matching in a graph G is a set of edges of which no pair shares an endpoint.

Definition 2.92. Let G be a graph with at least one edge. A set

$$S \subseteq E$$

is a matching if S is an independent set in $L(G)$.

We denote a matching of maximal cardinality in G by

$$\nu(G) = \alpha(L(G)).$$

A matching of maximal cardinality is called a maximum matching.

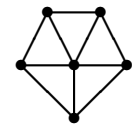


Figure 2.12: The 5-wheel

Exercise 2.57

Check Definition 2.92 with the text above it.

ν is the 13th Greek alphabet letter 'nu.' In the English alphabet, the 13th letter is m, for matching.

2.15 Independent Set in Claw - Free Graphs

Figure 2.13: The claw

CLAW - FREE GRAPHS GENERALIZE LINEGRAPHS. Of course they can be recognized in polynomial time. Minty designed a polynomial – time algorithm to find maximum independent sets in claw – free graphs. We describe this algorithm. Since linegraphs are claw – free, this implies that there is a polynomial – time algorithm to find a maximum matching in a graph.

When $\mathcal{F}$ is some finite collection of graphs, then the class of $\mathcal{F}$-free graphs is the set of those graphs that have no induced subgraph isomorphic to an element of $\mathcal{F}$.

BTW Harary exhibits a complete list of nine forbidden induced subgraphs that characterize linegraphs. (For example — the 5-wheel is claw-free but not a linegraph).

2.15.1 The Blossom Algorithm

Let us first recapitulate Edmonds' algorithm to compute a maximum matching in a graph.

A chain is a sequence of vertices

$$P = [v_0 \quad \cdots \quad v_t]$$

such that successive elements are adjacent. Let M be a maximal matching. Let X be the set of vertices that are not in any line of M.

Berge showed that M is not maximal if and only if there exists an M-augmenting path — that is — a path that starts and ends with distinct points in X and whose edges alternate between M and $E \setminus M$. Berge's original proposal to find an augmenting path via a depth-first-search procedure did not work because the path could end up in an odd cycle. It was Edmonds' idea to shrink the cycle into one new point and start the search afresh.

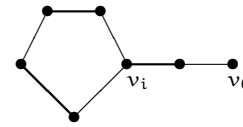


Figure 2.14: The figure shows a flower. The thick lines represent edges of M.

Definition 2.93. A chain $P = [v_0 \quad \cdots \quad v_t]$ is alternating if for each i exactly one of the edges $\{v_{i-1}, v_i\}$ and $\{v_i, v_{i+1}\}$ is in M.

We first show that we can find a shortest alternating chain with endpoints in X. Let A be the set of edges of a directed graph (V, A) defined as follows

$$A = \{u \to v \mid \exists_{x \in V} \quad \{u, x\} \in E \quad \text{and} \quad \{x, v\} \in M\}.$$

Then each alternating chain which starts and ends in X is a directed path from X to $N_G(X)$ in (V, A). Thus we can find an alternating chain in G in polynomial time.

Definition 2.94. An alternating chain

$$P = [v_0 \quad \cdots \quad v_t]$$

is a flower if

1. t is odd
2. $v_0, \ldots, v_{t-1}$ are distinct
3. $v_t = v_i$ for some $i < t$ where i is even.

The circuit $[v_i, \ldots, v_t]$ is called the blossom of the flower.

Lemma 2.95. *A shortest alternating chain*

$$P = [v_0 \quad \cdots \quad v_t]$$

from X *to* X *is either an augmenting path or* $[v_0 \quad \cdots \quad v_j]$ *is a flower for some* $j \leqslant t$.

Proof. Assume P is not a path. Let $i < j$ be such that $v_i = v_j$ and such that j is as small as possible. Thus $v_0, \ldots, v_{j-1}$ are distinct.

If $j - i$ is even delete $v_{i+1}, \ldots, v_j$ from P and obtain a shorter alternating chain from X to X.

Assume $j - i$ is odd. The case where j is even and i is odd cannot occur since this would imply that two edges of the matching meet in v_i.

Thus we may assume that j is odd and i is even which implies that

$$[v_0 \quad \cdots \quad v_j]$$

is a flower. □

Let B the the blossom of a flower. The graph G/B replaces the set of vertices of B by one new vertex. We call this new vertex B. The edges of the graph G/B are

$$E(G/B) = \{\{x,y\} \mid \{x,y\} \in E \quad \text{and} \quad x,y \notin B\} \cup \{\{B, y\} \mid y \in N_G(B)\}. \quad (2.43)$$

For the matching M in G we let M/B be the set of corresponding edges in G/B. An edge of M with both ends in B is not an edge of M/B.

Lemma 2.96. *Let*

$$B = [v_i \quad \cdots \quad v_t]$$

be a blossom in G. *Then* M *is a matching of maximal cardinality in the graph* G *if and only if* M/B *is a matching of maximal cardinality in the graph* G/B.

Proof. Assume that M/B is not maximum in G/B. Let P be an augmenting path. If P does not contain B then it is an augmenting path in G.

Assume that P enters B by an edge

$$\{u, B\} \notin M/B.$$

Thus $\{u, v_j\} \in E$ for some $j \in \{i, \ldots, t\}$. If j is odd then replace B in P by $[v_j\, v_{j+1} \quad \cdots \quad v_t]$. If j is even then replace B by $[v_j\, v_{j-1} \quad \cdots \quad v_i]$. In both cases we obtain an augmenting path in G.

Now assume that $|M|$ is not maximal. We may assume that $i = 0$ — that is — $v_i \in X$. Otherwise we can replace M by the symmetric difference $M \div Q$ where Q is the set of lines in the chain $[v_0 \quad \cdots \quad v_i]$.

Let $P = [u_0 \quad \cdots \quad u_s]$ be an augmenting path in G. When P does not visit B then P is an augmenting path in G/B. If P visits B then we may assume that $u_0 \notin B$ since otherwise we can replace P by its reverse. Let u_j be the first vertex of P in B. Then $[u_0 \quad \cdots \quad u_{j-1}\, B]$ is an augmenting path in G/B. — Thus — $|M/B|$ is not maximal. □

Edmonds' algorithm can be implemented to run in $O(n^2 \cdot m)$ time. Micali and Vazirani show that a maximum matching can be computed in $O(\sqrt{n} \cdot m)$ time.

2.15.2 Minty's Algorithm

We assume that the reader is familar with Edmonds' algorithm to find a maximum matching in graphs. Minty's algorithm computes $\alpha(G)$ in claw-free graphs by reducing it to that of finding a maximum matching in an auxiliary graph constructed from the input graph.

Let G be claw-free and let B be a maximal independent set in G. Color the vertices of B black and the others white. Notice that every white vertex has 1 or 2 black neighbors.

An augmenting path is a path that runs between two white vertices — that each have only one black neighbor — and of which the white vertices form an independent set.

Lemma 2.97. *When two white vertices — both having two black neighbors — are adjacent then they have a common black neighbor.*

Proof. Let x and y be two white vertices that are adjacent. Notice that

$$|\,(\,N(x) \cup N(y)\,) \cap B\,| \geqslant 4$$

implies a claw with one of the two whites as a center (since the black vertices form an independent set). This is a contradiction. □

If there exists an independent set of cardinality larger than $|B|$ then there must exist an augmenting path. Minty's algorithm finds an augmenting path — if it exists — as follows.

Definition 2.98. A wing is a nonempty subset of white vertices that is a single white vertex that has only one black neighbor or the common neighborhood of two black vertices.

We refer to the wings of the second kind as the 'tipped wings.'

Definition 2.99. A black vertex is regular if either

1. it is incident with a white vertex that has only one black neighbor
2. it is incident with at least three tipped wings.

Lemma 2.100. *Let* b *be a regular black vertex of the second kind. Let* $p, q, r \in N(b)$ *and let them be in different tipped wings incident with* b*. Then the number of pairs in* $\{p, q, r\}$ *that are edges is odd.*

Proof. The graph would have a claw when the number of edges in $\{p, q, r\}$ were 0 or 2 (either with b as a center or the one of p, q and r that is adjacent to the other two). □

2.15.3 A Cute Lemma

In an attempt to find an augmenting path we may try to find an augmenting path that runs between a fixed pair of white vertices that have one black neighbor. By trying all $O(n^2)$ feasible pairs this will find an augmenting path if it exists.

This approach has the advantage that we can reduce the graph: Let s and t be two white vertices that are not adjacent and that have each exactly one black neighbor. In the search for an augmenting $s \sim t$-path we can remove all white neighbors of s and t and all other white vertices that have only one black neighbor. We refer to this graph as the reduced structure.

Minty first shows that the neighborhood of any regular back vertex has a partition into two classes such that all nonedges run between vertices in different classes. For a regular vertex b that is adjacent to s or to t one of the two classes is s or t and the other class is $N(b) \setminus \{s, t\}$. The following lemma concerns itself with the regular vertices that are incident with at least three tipped wings.

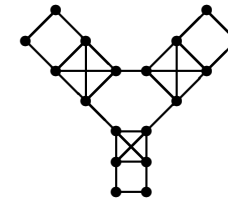


Figure 2.15: A claw-free graph; this one is a domino: every vertex is in two maximal cliques.

Lemma 2.101. *Let* b *be a black vertex that is incident with at least three tipped wings. There exists a unique partition of* $N(b)$ *into at most two parts such that all nonedges that run between whites in different wings have their endpoints in different parts and all edges between whites in different wings have their endpoints in similar parts.*

Proof. Notice that if x, y and z are three white vertices in different tipped wings at b then the number of edges among $\{x, y, z\}$ is odd. We call a triple $\{x, y, z\}$ with an odd number of edges among them an odd triple.

First notice that — when each triple in a graph is odd — then the graph is a union of at most two cliques. In that case we are done.

Consider a wing S and let $z \notin S$. Define a binary relation L_z on S as follows. Any two elements x and y in S are related if and only if $\{x, z\}$ and $\{y, z\}$ are both edges or both nonedges.

It is not difficult to check that any $z' \notin S$ produces the same relation $L_{z'} = L_z$ on S — that is — we can write L instead of L_z.

Make two vertices in a wing S adjacent when they are related in L and nonadjacent when they are not related in L. [48] Then every triple in the wing is an odd triple.

It is now easy to check that any triple $\{x, y, z\}$ in $N(b)$ is odd. It follows that $N(b)$ has a partition as claimed in the lemma. □

[48] Notice that any augmenting path can use only one white vertex in a wing. So the graph on the white vertices within a wing is of no importance. The lemma makes no claim on the edges that run between whites that are in the same wing.

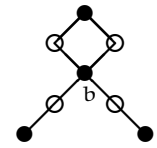


Figure 2.16: Three wings (edges with white endpoints are not shown)

Exercise 2.58

Let G be a graph and let Ω be its collection of odd triples. A Seidel switch with respect to some set $S \subseteq V$ changes all edges with one endpoint in S into nonedges and it changes all nonedges with one endpoint in S into edges.
Prove that a Seidel switch does not change Ω. When is a collection of triples the set of odd triples of a graph?
Hint: A collection of triples is the collection of odd triples in a graph if and only if every four elements contain an even number of even triples.

2.15.4 Edmonds' Graph

Notice that we lack a partition — into nonadjacent classes — of the neighborhoods of black vertices whose neighborhoods consists of two tipped wings. Minty calls these vertices irregular.

As a subroutine the algorithm determines whether two regular black vertices — say $\mathfrak{a}$ and $\mathfrak{b}$ — are connected by an alternating path whose white vertices form an independent set.

Consider a sequence $(W_0, \mathfrak{b}_0, W_1, \mathfrak{b}_1, \ldots, W_n, \mathfrak{b}_n, W_{n+1})$ where the $\mathfrak{b}_i$ are irregular black vertices and W_i is the wing at $\mathfrak{b}_i$ with tip $\mathfrak{b}_{i+1}$. We may assume that this sequence is maximal — that is — W_0 and W_{n+1} are wings incident with regular vertices. By dynamic programming we can determine whether there exists a path

$$[\mathfrak{a} \quad w_0 \quad \mathfrak{b}_0 \quad w_1 \quad \mathfrak{b}_1 \quad \cdots \quad w_{n+1} \quad \mathfrak{b}]$$

such that

1. $\mathfrak{a}$ and $\mathfrak{b}$ are the tips of W_0 and W_{n+1}
2. $\mathfrak{a}$ is not adjacent to $w_0 \in W_0$ and $\mathfrak{b}$ is not adjacent to $w_{n+1} \in W_{n+1}$
3. consecutive white vertices $w_i \in W_i$ and $w_{i+1} \in W_{i+1}$ are nonadjacent.

We refer to such an $\mathfrak{a} \sim \mathfrak{b}$-path as an irregular path.

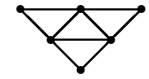


Figure 2.17: This is the sun. The sun is claw-free. It is the linegraph of the net.

Let N be the number of regular vertices in the graph. Edmonds' graph consists of a matching with N edges, representing the regular vertices. The endpoints of an edge represent the two classes of the partition of the regular black vertex. The graph has two more vertices s and t. These are joined by an edge to the node-classes of the two unique regular black neighbors.

By the subroutine described above for any two classes of regular black vertices we can decide whether they are connected by an augmenting path that uses no irregular vertices. Between any two of the $2N$ endpoints of edges in the matching add an edge if the two regular black vertices are connected by an irregular path that uses the two classes.

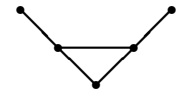


Figure 2.18: This graph is called the bull. The bull is claw-free, but has an edge-contraction that produces a claw. Thus, the class of claw-free graphs, is not closed under edge-contractions. BTW, what is the complement of the bull?

Lemma 2.102. *There exists an augmenting $s \sim t$ – path if and only if Edmonds' graph has an augmenting path.*

We leave it as an exercise to check the correctness.

Theorem 2.103. *There exists an* $O(n^5)$ *algorithm that computes a maximum independent set in claw-free graphs.*

Proof. The problem reduces to finding a augmenting path in Edmonds' graph. The existence of irregular paths can be computed in (overall) $O(n^2)$ time. Finding an augmenting path in Edmonds' graph can be done in $O(n^3)$ time via the blossom algorithm. Since the algorithm tries all feasible pairs s and t as endpoints of an augmenting path the algorithm runs in $O(n^5)$ time. □

Exercise 2.59

The problem to find $\alpha(G)$ in a triangle-free graph is NP-complete. Reduce this problem to the clique problem in claw-free graphs. Thus, finding $\omega(G)$ in claw-free graphs is NP-complete.

Faenza et al. show that the independence number in claw-free graphs can be computed in $O(n^3)$ time.

2.16 Dominoes

A natural generalization of the class of linegraphs of bipartite graphs, is the class of dominoes.

Definition 2.104. A graph is a domino if every vertex is in at most two maximal cliques.

Notice that linegraphs of bipartite graphs are dominoes. [49] Not all linegraphs are dominoes — for example — the linegraph of the diamond is the 4-wheel W_4 and W_4 is not a domino.

[49] I spell: "one domino" and "two dominoes."

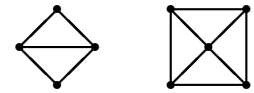


Figure 2.19: The linegraph of the diamond, on the left, is the 4-wheel, on the right. The 4-wheel is not a domino.

Exercise 2.60

Show that every domino has at most n maximal cliques.

Exercise 2.61

Show that the class of dominoes is hereditary — that is — a graph G is a domino if and only if every induced subgraph of G is a domino.

Dominoes can be characterized in various ways.

Theorem 2.105. *The following propositions are equivalent.*

1. G *is a domino*
2. G *is* $\{W_4, \mathsf{claw}, \mathsf{gem}\}$*–free.*

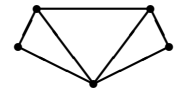


Figure 2.20: The gem

Remark 2.106. The class of dominoes can be recognized in linear time. — That is — there is a linear–time algorithm that checks whether a graph is a domino. The algorithms operates by identifying vertices that have the same closed neighborhood. The graph on the equivalence classes is a linegraph (with some additional properties). — Actually — a graph is a domino if and only if its representative is the linegraph of a triangle-free graph in which every vertex is adjacent to at most one pendant vertex.

Exercise 2.62

Let G be a graph. Call two vertices equivalent if they have the same closed neighborhood. Show that this defines an equivalence relation on $V(G)$. The representative $R(G)$ of a graph G is the graph with

$$V(R) = \{ X \mid X \subseteq V(G) \text{ is an equivalence class } \}.$$

Two vertices of R are adjacent if a pair of elements of the classes are adjacent. Design a linear–time algorithm that computes the representative of a graph.

A binary relation is an equivalence relation if it is reflexive, symmetric, and transitive. To be precise, a binary relation $\sim$ is an equivalence relation if $\sim$ satisfies

1. $\forall_x\ x \sim x$,
2. $\forall_x \forall_y\ x \sim y \Rightarrow y \sim x$,
3. $\forall_x \forall_y \forall_z\ (x \sim y \text{ and } y \sim z) \Rightarrow x \sim z$.

The sets of mutually equivalent vertices are called equivalence classes.

Exercise 2.63

A graph is strongly regular if there are numbers k, λ and μ such that

(i) all vertices have the same degree — that is — the graph is regular with degree k:

$$\forall_{x \in V} \quad d(x) = |N(x)| = k$$

(ii)

$$\forall_{x \in V} \quad \forall_{y \in V} \quad x \neq y \quad \Rightarrow$$
$$|N(x) \cap N(y)| = \begin{cases} \lambda & \text{if } \{x, y\} \in E \\ \mu & \text{if } \{x, y\} \notin E. \end{cases}$$

— For example — the Petersen graph is strongly regular with parameters

$$(n, k, \lambda, \mu) = (10, 3, 0, 1).$$

(I) Prove that $L(K_n)$ is strongly regular. Is it a domino?

(II) Prove that the adjacancey matrix of a strongly regular graph satisfies

$$A^2 = k \cdot I + \lambda \cdot A + \mu \cdot (J - I - A)$$

where I is the identity matrix and J is the matrix with all elements equal to 1.

HINT: Notice that $J - I - A$ is the adjacency matrix of $\bar{G}$.

2.17 Triangle partition of planar graphs

Definition 2.107. A graph has a partition if its set of edges can be partitioned into triangles. [50] [51]

In this chapter we show that there is a linear - time algorithm to check if a planar graph has a partition.

[50] A triangle is a clique in the graph with 3 vertices.

[51] Finding a minimum set of triangles that COVERS the edges of a planar graph is NP-complete.

Definition 2.108. Let G be a plane graph — that is — the graph G is planar and is given with an embedding in the plane. A triangle T partitions the plane in two open regions say 'inside' and 'outside.' If both regions contain vertices of G the triangle is separating.

Definition 2.109. Let T be a separating triangle and let $x \in V(T)$. The inside degree of x — say $d(x)$ — is the number of edges that is incident with x and some vertex inside T.
A separating triangle is even if $d(x)$ is even for every $x \in V(T)$.

The dual

Exercise 2.64

Assume that G has an edge which is in only one triangle. Say the triangle has edges e_1, e_2 and e_3. Then G has a partition if and only if $G - \{e_1, e_2, e_3\}$ has a partition.

Exercise 2.65

Show that there is a linear-time reduction to the case where the graph G is biconnected — that is — henceforth we assume that G is biconnected. Furthermore we may assume that every vertex of G has at least three neighbors. [52]

[52] A graph is biconnected if each minimal separator has at least two vertices.

Lemma 2.110. *Let H be the dual of G. If G has a partition then H is bipartite.*

Proof. Assume G has a partition. Let C be a cycle in H. The set of edges of C is a cut in G. Every triangle of the partition has either all its vertices on the same side of the cut or one vertex on one side and two on the other side. This shows that the cut has an even number of edges — and so C is even. □

The triangle partition algorithm

— By now — we may assume the following. [53]

[53] Check !

1. G is biconnected
2. the dual H is bipartite
3. every vertex of G has even degree at least 4
4. every edge of G is in at least two triangles.

Graphs without separating triangles

If the graph has no separating triangles then every triangle is a face.

Lemma 2.111. *Assume* G *has no separating triangle. Then* G *has a partition if and only if every vertex of one color class of* H *has degree 3.*

Proof. Let H_1 be a color class of H and assume that all vertices of H_1 have degree 3. Then the vertices of H_1 form a partition of $E(G)$ into triangles.

Assume that G has a partition. The triangles of the partition are faces and the corresponding vertices in H have degree 3. Between any two of them the distance is even so they form a color class of H. □

Graphs with separating triangles

Let $\mathcal{P}$ be a partition of $E(G)$ into triangles. A separating triangle $S = \{x, y, z\}$ is of one of the following types.

Type 1. $S \in \mathcal{P}$ or the three edges of S are in triangles with the third vertex inside S

Type 2. the three edges of S are in triangles of $\mathcal{P}$ with the third vertex outside S

Type 3. some edge of S is in a triangle of $\mathcal{P}$ with the third vertex inside S and some edge of S is in a triangle of $\mathcal{P}$ with the third vertex outside S.

Exercise 2.66

If a separating triangle S is even then it is of Type 1 or Type 2 in any partition.

HINT: Let S be a separating triangle and let $\mathcal{P}$ ve a partition of the edges of G into triangles. Let G' be the graph induced by S and the vertices inside S. When $S \in \mathcal{P}$ then S is even (otherwise G' has no partition).

Let $\{x, y\} \in E(S)$ and assume that $\{x, y\}$ is in a triangle of $\mathcal{P}$ with the third vertex outside S. Assume that the two other edges of S are in triangle with their third vertex inside S. (So S is of Type 3.)

Remove the edge $\{x, y\}$ from the graph G'. There is a partition of the edges of $G' - \{x, y\}$ into triangles — and so — the degree of x and y must be even in $G' - \{x, y\}$. But then S is not an even triangle in G.

Exercise 2.67

All even separating triangles can be found in linear time.

HINT: Use Baker's method to partition $V(G)$ into layers. (See Theorem 4.246 on Page 323.) [54]

[54] Jiawei Gao, Ton Kloks and Sheung-Hung Poon, *Triangle - partitioning edges of planar graphs, toroidal graphs, and* k*-planar graphs*, Springer - Verlag, Lecture Notes in Computer Science 7748 (2013), pp. 194–205.

Definition 2.112. A separating triangle is outermost if none of its vertices is inside any other separating triangle.

Consider the graph G^* obtained from G by removing the interior of all outermost even separating triangles. So the graph G^* has no even separating triangles. A special region of G^* is a face that is an outermost even separating triangle in G.

Let $\mathcal{P}$ be a partition of G^*. Every even region or face is of one of two types.

Type a. the special region or face is a triangle of $\mathcal{P}$

Type b. the edges of the boundary are in triangles of $\mathcal{P}$ that have their third vertex outside.

Lemma 2.113. *Assume* G^* *has a partition. Let* H_1 *and* H_2 *be the two color classes of the dual. Then all vertices of* H_1 *are of Type a and all vertices of* H_2 *are of Type b or vice versa.*

Proof. Along any path in the dual of G^* the types of the vertices must alternate. The dual is connected so all vertices of one color class are of the same type.

This proves the lemma. □

Theorem 2.114. *There exists a linear - time algorithm to find a partition of the edges of a planar graph in triangles.*

Proof. There are only two ways to partition G^*.

Let S be an outermost even separating triangle. If S is labeled Type a then a recursive step checks if the inside including S has a partition. If S is labeled as Type b then the interior (without S) is processed in a recursive step.

A list of all even separating triangles can be found in linear time. All recursive steps are performed on separate subgraphs of G. This proves that the algorithm runs in linear time.

This proves the theorem. □

2.17.1 Intermezzo: PQ *- trees*

Booth and Luecker introduced PQ - trees in 1976 as a data - structure that is useful for the recognition of eg planar graphs and interval graphs.

Implementations of PQ - trees allow linear - time recognition algorithms for interval graphs and for planar graphs.

Let V be a finite set. A PQ - tree is a rooted tree T and a bijection from the elements of V to the leaves of T. Each internal node is labeled P or Q and has at least two children.

The PQ - tree represents a set of permutations of V:

1. the children of a P - node can be re-ordered in any way
2. the order of the children of a Q - node can be reversed (and that is the only other valid order).

Let G be an interval graph — that is — there is an ordering of the maximal cliques in G

$$C_1 \quad \cdots \quad C_t$$

such that each vertex is contained in a consecutive subset.

Consider the $(0,1)$ - matrix with V as its columns and the maximal cliques in G as its rows and that has a 1 precisely when a vertex is in a clique. Then the rows can be permuted so that all the ones in a column are consecutive.

Exercise 2.68

Design an algorithm:
INPUT: A graph G and the set of all the maximal cliques in G — say — $\{C_1, \cdots, C_t\}$.
OUTPUT: A permutation of the cliques such that the (clique - vertex) - incidence matrix has all ones in each column consecutive.

HINT: Build a PQ - tree. Start with a tree that has all its leaves adjacent to the root and identified with $C_1, \cdots C_t$. Label the root as a P - node. Add the vertices one by one and rebuild the tree to satisfy the consecutive ones property.

2.18 Games

IT'S ALMOST CHRISTMAS — LET'S PLAY SOME GAMES!

One of my teachers used to say: "Only games played between two people are interesting. Larger groups of players give rise to fights and 1-player games are boring!"

2.18.1 Snake

SNAKE IS A THRILLING GAME played on a graph as follows. — Of course — the game is played between two players. The 'game - board' is a graph G. The two players take turns in making (legal) moves. The player who can't make a legal move loses the game.

Johan Cruijff was once the greatest football player in Holland. One of his proverbs was: "Football is a game of mistakes. Whoever makes the fewest wins!"

The two players together build a path in the graph. One endpoint of the path is the 'head' and the only legal moves that are available to a player (when it is his turn) extend the path with one vertex that is adjacent to the head. [55] A newly added vertex becomes the head of the path.

[55] Of course, only legal moves are allowed. A player is allowed to make a legal move when it is his turn.

Player 1 makes the first move and he chooses one point in the graph to start the path.

Exercise 2.69

Show that player 1 can easily win the game if the graph has an isolated vertex.

C. Berge, *Combinatorial games on a graph*, Discrete Mathematics **151** (1996), pp. 59–65.

We are interested in the question whether player 1 has a winning strategy.

Snake was invented by Berge and he formulated the following beautiful theorem in 1996.

Theorem 2.115. *Player 1 wins the game snake if and only if the graph has no perfect matching.*

A perfect matching in a graph is a matching with $n/2$ edges; so it covers all the vertices.

Proof. Suppose G has a perfect matching. We show that player 2 has a winning strategy.

To win the game player 2 fixes a perfect matching M and carries that in his head. Player 1 chooses a vertex x to start the game. There is a unique edge $e \in M$ that contains x. Player 2 chooses the other endpoint of e to extend the path.

At any point in the game when player 1 enters a new edge of M player 2 chooses the other end of that edge. Since M is perfect this is a winning strategy for player 2.

The exercise below takes care of the converse.

This proves the theorem. □

Exercise 2.70

Suppose that G has no perfect matching. Show that player 1 has a winning strategy.

2.18.2 Grundy values

When it is a player's turn he is faced with a position in the game. Assume that the positions in a game form a DAG when there is an arc from position x to position y if y is reached from x in one move (by either player). [56]

[56] We want to consider only finite games; if the digraph has a directed cycle a game can go on forever.

Give each position a Grundy value defined as follows. Each sink in the DAG has Grundy value 0. Let x be a position with some outgoing arcs. The Grundy value of x is the smallest element in $\mathbb{N} \cup \{0\}$ that is not a Grundy value of any of its out-neighbors.

A sink in the DAG is a position that ends the game.

Let s denote the starting - position — ie — s is the position in which player 1 has to make his first move. Call the Grundy value of s the Grundy value of the game.

My teacher once said that we could try to simulate chess with a DAG; but it's made hard by rules that involve the history of the game (like 'castling'). "And then — you need to deal with a case where you offer a draw and your opponent gets a red face and resigns!"

Player 1 has a winning strategy if and only if the Grundy value of the game is not 0.

To see that assume that the Grundy value of s is not 0. By the definition of the Grundy value there must be a position y that player 1 can reach which has Grundy value 0. Player 1 makes that move. Since the Grundy value of y is zero player 2 can only make moves to positions that have a non-zero Grundy value. — So — the game ends when player 1 makes the final move to a sink.

A natural question is whether player 1 has a winning strategy.

Input: A graph G and a game (ie a rule which defines legal moves).

Output: Decide whether the Grundy value of a the game is 0.

For example, there is a polynomial-time algorithm to decide if the Grundy value of SNAKE is zero; just figure out if the graph has a perfect matching.

2.18.3 De Bruijn's game

De Bruijn shows that is can be quite difficult to find a winning move.

To decide a game is to decide whether its Grundy value is 0. If the Grundy value is > 0 then player 1 has a winning move. In this section we show a game in which it is not easy to find that winning move.

Consider this game played on the set $[n]$. During the game two players alternate and scratch out certain numbers of $[n]$. A number can be scratched out only once and when no number is left the player who has to make a move loses the game.

The rule to scratch out numbers is this. When it is a player's turn he chooses a number that has not been scratched out before. The move scratches out this number and all its divisors (ie those divisors that were not scratched out already).

So the number 1 gets scratched out at the first move.

PLAYER 1 HAS A WINNING STRATEGY.

To see that suppose player 1 chooses 1. This does not really change the game — that is — if now player 2 wins the game then player 1 could have made that winning move instead of playing 1! — In other words — player 1 wins the game.

In this game player 1 has a 'waiting move.'

Player 1 wins the game but finding the winning move is not feasible —say — when $n > 100$.

Exercise 2.71

The NIM - game is played with some piles of stones. When it is a player's turn he must choose one nonempty pile and remove some stones from it. When no pile has any stone left the player that has to make a move loses the game.

Let there be k piles and let the number of stones in these piles be

$$n_1 \quad \cdots \quad n_k.$$

The Grundy value of this game is the nim - sum

$$n_1 \quad \oplus \quad \cdots \quad \oplus \quad n_k.$$

To obtain the nim - sum add the n_i with the following addition rule

$$\begin{aligned} a \oplus b \quad = \quad & \min\{\, k \in \mathbb{N} \cup \{0\} \mid \forall_{a' < a} \quad \forall_{b' < b} \\ & k \neq a' \oplus b \quad \text{and} \quad k \neq a \oplus b' \,\}. \end{aligned}$$

Another way to obtain $a \oplus b$ is to write a and b in binary and then to add them up bit - by - bit without using a carry.

Exercise 2.72

In this game the board is a forest F of k rooted trees. On the root of each tree lies a coin. A legal move chooses one tree in the forest and moves the coin in this tree to a point that is further away from the root.

Let d_i denote the maximal distance of any point in $T_i \in F$ from the root. The Grundy number of this game is

$$d_1 \quad \oplus \quad \cdots \quad \oplus \quad d_k.$$

2.18.4 Poset games

Poset games are played on a poset. Two players play a game on a poset $(P, \leqslant)$. When it is his turn a player selects an element x of P; this removes x and all elements $y \geqslant x$.

Example 2.116. 1. Clearly NIM is a poset game: the poset is a union of chains (the piles).

2. HACKENDOT is a game played on a forest of rooted tree. The selection as a move of a vertex removes all vertices that are on the path from the vertex to a root. (When the forest is a tree then player 1 wins.)

J. Úlelha, A complete analysis of Von Neumann's Hackendot, *International Journal of Game Theory* **9** (1980), pp. 107–113.

Deuber and Tomassé show that the Grundy value of an N - free poset - game can be computed in $\mathsf{O}(n^4)$ time.

W. Deuber and S. Thomassé, Grundy sets of partial orders. Technical report, University Bielefeld, 1980.

2.18.5 Coin - turning games

LET A BOARD BE $[n]$ WITH A COIN ON EVERY ELEMENT THAT IS SHOWING head OR tail. A move is a turn of two coins subject to the condition that the right - most coin of the two turns from head to tail.

Write the numbers $1 \cdots n$ from left to right.

As usual, when a player can't make a legal move he loses the game.

Exercise 2.73

Suppose there is only one coin that is showing head. Show that the game is equivalent to NIM with one pile of stones. How many stones are there in the pile of the NIM-game?

HINT: If there is only one coin that shows head then a legal move is the same as 'shifting the head to the left.' So, it is equivalent to NIM with one pile of stones. The number of stones on the pile in NIM is the number of positions the head can move to the left; if it is in position k then it can move $k-1$ positions to the left. (If the head is in position 1 player 1 loses the game; accordingly $g(1) = 0$.)

LET'S TAKE A BOLD STEP and see what happens if we simulate the coin - turning game by a NIM - game that has one pile for every head in the set.

Let $A \subseteq [n]$ be the set of elements where the coin is showing head. Simulate this game by a NIM - game; with a pile of stones for every $a \in A$. The number of stones in a pile — say $g(a)$ — is the number of positions it can move to the left.

Claim: the two games have the same Grundy value:

$$\bigoplus_{a \in A} g(a).$$

Proof. A difficulty arises only when two coins are turned that are both head. In a 'perfect' simulation of the coin - turning game the two piles would 'disappear.' The two piles in the NIM - game become of equal size — that is — they have the same number of stones and so the sum of their Grundy values equals 0.

This proves the claim. □

Exercise 2.74

Suppose the coin - turning game is played on an $n \times m$ - grid. Call (n, m) - corner of the grid the North - East. One each point of the grid lies a coin which shows head or tail.

A legal move turns all coins of a subgrid subject to the condition that the North - East - corner of the subgrid turns from head to tail.

Show that the Grundy value can be expressed as

$$\bigoplus_{a \in A} g(a), \tag{2.44}$$

where A is the set of vertices of the grid on which the coin shows head. (What is $g(a)$?)

Exercise 2.75

Let $(P, \preceq)$ be a poset. Define the coin - turning game analogously to the above. Show that there is an efficient way to decide if player 1 wins this game.

Hint: Introduce a turning set for each element in the poset. All coins in a turning set $T(a)$ turn when a player turns the coin in a. In this game we assume that each element a is the unique maximal element of its turning set $T(a)$.

A legal move flips all coins of a turning set $T(a)$ provided that the coin in a shows head. Show that the Grundy value of the game can be expressed as in (2.44).

2.18.6 Nim - *multiplication*

H. W. Lenstra, Jr., NIM - multiplication. Technical Report Institute des Hautes Etudes Scientifiques IHES/M/78/211. Research supported by the Netherlands Organization for the Advancement of Pure Research (Z.W.O), 1978.

This paper defines a game as follows.

This definition should end with an exclamation mark!

Definition 2.117. A game is a set.

To explain this definition a game is identified with its initial position and a position is a set of options. Each option is again a position. — So — an element of a game is a position (a set of options) and every element of a position is again a position. In this section all elements of a set are sets.

To play a set S Player 1 chooses an element of S — say — S'. Then Player two chooses an element S'' of S' and so on. The player who needs to choose an element from an empty set loses the game. We assume that this will — eventually — occur, after a finite number of moves.

While we're at it let's define the Grundy number of a game A as $g(\varnothing) = 0$ and

$$g(A) \quad = \quad \min\{\, k \mid k \neq g(\ell) \quad \text{for} \quad \ell \in A \,\}$$

Then player 1 has a winning strategy if and only if the Grundy value of the game is not zero.

The sum of two game A and B is the game

$$A + B \quad = \quad \{\, a + B,\ b + A \mid a \in A \quad \text{and} \quad b \in B \,\}$$

We assume that you are familiar with the sum - theorem.

Theorem 2.118 (The sum theorem)**.** *The Grundy value of the sum of two games* $A + B$ *is*

$$g(A + B) \quad = \quad g(A) \quad \oplus \quad g(B),$$

where $\oplus$ *is the* NIM *- addition:*

$$\alpha \oplus \beta \quad = \quad \min\{\, k \mid k \neq \alpha' \oplus \beta \quad \textit{and} \quad k \neq \alpha \oplus \beta' \quad \textit{for all } \alpha' < \alpha \textit{ and } \beta' < \beta \,.\,\}.$$

Let us define the product of two games as

$$A \times B \quad = \quad \{\, (a \times B) + (A \times b) + (a \times b) \mid a \in A \quad \text{and} \quad b \in B \,\}.$$

Then the Grundy value of the product is

$$g(A \times B) \quad = \quad g(A) \circ g(B)$$

where the product $n \circ m$ of two numbers is the smallest number different from $(n' \circ m) \oplus (n \circ m') \oplus (n' \circ m')$ for all $n' < n$ and $m' < m$. (2.45)

We want no zero divisors:

$(n - n') \circ (m - m') \neq 0.$

So the Grundy value of $n \circ m$ is the smallest number different from all

$(n' \circ m) \oplus (n \circ m') \oplus (n' \circ m').$

Suppose we play the product game with two naturural numbers n and m. After t moves the position looks like

$$(a_1 \circ b_1) + (a_2 \circ b_2) + \cdots + (a_{2t+1} \circ b_{2t+1}).$$

For any pair a, b the term $(a \circ b)$ may appear many times but only the parity of the number of occurrences of $(a \circ b)$ is of interest.

A legal move is to replace a pair — say (a, b) — with three pairs

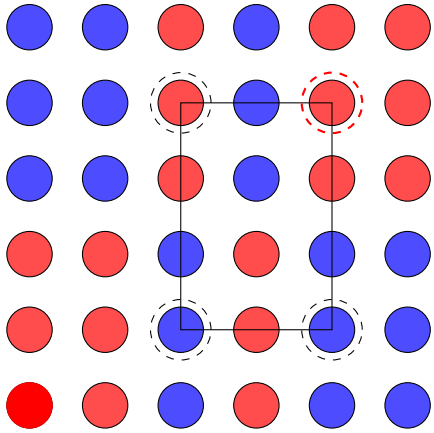
$$(a' \circ b) + (a \circ b') + (a' \circ b'),$$

where $a' < a$ and $b' < b$.

$(\mathbb{N}, \oplus, \circ)$ is a field of characteristic two.

If one of the three new term is already in the product then the two equal terms cancel each other out because the sum of two equal games is zero.

We can represent the positions of the $n \circ m$ - game by a rectangular $n \times m$ - grid. In each point of the grid there is a coin showing red or blue (head or tail).



The rule for a legal move is as follows. Choose a rectangle of which the North - East corner is red. Switch the color at the 4 corners of the rectangle.

Theorem 2.119 (The product theorem)**.** *The Grundy number of a product game is the sum over all the red nodes of the nim - product of the two coordinates.*

In his paper (Exercise 4) Lenstra describes an algorithm to calculate the NIM - product of two numbers $n \circ m$.

2.18.7 P_3 - *Games*

Definition 2.120. Let G be a graph. A set $S \subseteq V$ is P_3 - convex if

$$\forall_{x \notin S} \quad |N(x) \cap S| \quad < \quad 2.$$

Notice that this defines an alignment — that is

1. $\varnothing$ and V are P_3-convex
2. if A and B are P_3-convex then $A \cap B$ is P_3-convex.

Exercise 2.76

Let $\mathcal{L}$ be the collection of P_3-convex sets. Define the hull operator $\sigma : 2^V \to \mathcal{L}$ by

$$\sigma(A) \quad = \quad \text{the smallest } P_3\text{-convex set that contains } A.$$

Show that this is a proper definition.

> Two players play the P_3-game. The board is a graph. In a move certain vertices of the graph get labeled. Initially the set of labeled vertices $S = \varnothing$.
>
> A player selects an unlabeled vertex x. This changes the set of labeled vertices as follows.
>
> $$S \quad \leftarrow \quad \sigma(S + x)$$
>
> (The move labels all vertices of $\sigma(S + x)$.)

(At each point of the game prior to a move the set of labeled vertices S is P_3-convex.)

Exercise 2.77

Prove the following theorem.

Theorem 2.121. *There exists an* $O(n^2)$ *algorithm to decide the* P_3 *- game on paths.*

HINT: Compute the Grundy value of the game played on every subpath — using dynamic programming.

In the connected P_3 - game the moves are restricted so that the set of labeled vertices S must induce a connected subgraph.

Wing Kai Hon, Ton Kloks, Fu-Hong Liu, Hsiang-Hsuan Liu and Tao-Ming Wang, P_3 - *games*. Manuscript on arXiv: 1608.05169, 2016.

Exercise 2.78

1. Player 1 wins the connected P_3 - game on P_n if and only if
$$n \neq 2.$$
2. Player 1 wins the connected P_3 - game on the cycle C_n if and only if
$$n = 2 \mod 3.$$

Exercise 2.79

1. Show that there is a polynomial - time algorithm to decide the connected P_3 - game on trees.
2. Show that there is a polynomial - time algorithm to decide the P_3 - game on cographs.
3. The ladder is the Cartesian product $P_2 \times P_n$. Show that Player 1 wins the connected P_3 - game on the ladder if and only if
$$n = 0 \mod 6.$$

2.18.8 Chomp

It is also called 'the take-away game.'

TWO PLAYERS PLAY A GAME ON A GRAPH. The name of the game is CHOMP. When it is his turn a player removes a vertex or edge of the graph. [57] The game ends when there are no more vertices or edges left. The game ends when there is no graph left to play with.

[57] The removal of a vertex also removes all edges that are incident with it.

Exercise 2.80

Show that player 1 loses the game on a triangle. Show that the Grundy value for Chomp on K_n equals $n \bmod 3$.

Exercise 2.81

Show that there is an efficient way to decide the winner of a game of chomp on K_n^+ which consists of a clique with n vertices and one extra vertex that is adjacent to exactly one vertex in the clique.

A useful tool to compute the Grundy value of this game is the flipping lemma. A flip is an automorphism $\sigma : V(G) \to V(G)$ which satisfies

1. for every $x \in V$ $\{x, \sigma(x)\} \notin E$
2. $\sigma^2 = \sigma$.

The kernel of a flip is the set

$$\{x \mid \sigma(x) = x\}.$$

Lemma 2.122 (The flipping lemma)**.** *The Grundy value for chomp on* G *equals the Grundy value on any kernel of a flip.*

Kandhawit and Ye — extending older results of Draisma and Van Rijnswou for the Grundy value of forests — showed that for bipartite graphs the Grundy value equals

$$\phi(G) = n_2 + 2 \cdot m_2, \tag{2.46}$$

where n_2 and m_2 are the numbers of vertices and edges of the graph modulo two.

Exercise 2.82

1. Show that an even wheel has a flip with kernel P_3. — Consequently — even wheels have Grundy value 1.
2. Show that also odd wheels have Grundy value 1.

Exercise 2.83

Is there a polynomial time - algorithm to decide the winner of a game of chomp played on a cograph?

HINT: Every cograph is either one vertex or the join or the union of two smaller cographs. — Clearly — the Grundy value of the graph is the nim - sum of the Grundy values of its components.