Jagiellonian University
Department of Theoretical Computer Science

Adrian Siwiec

Perfect Graph Recognition and Coloring

Master Thesis

Supervisor: dr inż. Krzysztof Turowski

September 2020

Abstract

TODOa

List of definitions

Contents

Definitions

Run proofreader on all text (temporarily disabled because it slowed down IDE)

We use standard definitions, sourced from the book by Béla Bollobás *Modern graph theory*, modified and extended as needed.

Definition 0.1 (graph)**.** *A* graph G *is an ordered pair of disjoint sets* (V, E) *such that* E *is the subset of the set* $\binom{V}{2}$ *that is of unordered pairs of* V*.*

We will only consider finite graphs, that is V and E are always finite. If G is a graph, then $V = V(G)$ is the *vertex set* of G, and $E = E(G)$ is the *edge set*. The size of the vertex set of a graph G will be called the *cardinality* of G.

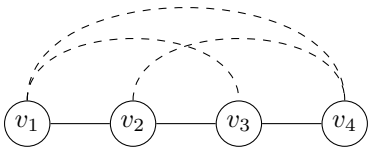


Figure 1: An example graph G_0

An edge $\{x, y\}$ is said to *join*, or be between vertices x and y and is denoted by xy. Thus xy and yx mean the same edge (all our graphs are *unordered*). If $xy \in E(G)$ then x and y are adjacent, connected or neighboring. By $N(x)$ we will denote the *neighborhood* of x, that is all vertices y such that xy is an edge. If $xy \notin E(G)$ then xy is a *nonedge* and x and y are *anticonnected*.

Figure shows an example of a graph $G_0 = (V, E)$ with $V = \{v_1, v_2, v_3, v_4\}$ and $E = \{v_1v_2, v_2v_3, v_3v_4\}$. We will mark edges as solid lines on figures. Nonedges significant to the ongoing reasoning will be marked as dashed lines.

Definition 0.2 (subgraph)**.** $G' = (V', E')$ *is a* subgraph *of* $G = (V, E)$ *if* $V' \subseteq V$ *and* $E' \subseteq E$*.*

Definition 0.3 (induced subgraph)**.** *If* $G' = (V', E')$ *is a subgraph of* G *and it contains* all edges *of* G *that join two vertices in* V'*, then* G' *is said to be* induced subgraph *of* G *and is denoted* $G[V']$*.*

Given a graph $G = (V, E)$ and a set set $X \subseteq V$ by $G \setminus X$ we will denote a induced subgraph $G[V \setminus X]$.

For example $(\{v_1, v_2, v_3\}, \{v_1v_2\})$ is *not* an induced subgraph of the example graph G_0, while $(\{v_1, v_2, v_3\}, \{v_1v_2, v_2v_3\}) = G_0[\{v_1, v_2, v_3\}] = G_0 \setminus \{v_0\}$ is.

Definition 0.4 (X-completeness)**.** *Given set $X \subseteq V$, vertex $v \notin X$ is* X-complete *if it is adjacent to every node $x \in X$. A set $Y \subseteq V$ is X-complete if $X \cap Y = \emptyset$ and every node $y \in Y$ is X-complete.*

Definition 0.5 (path)**.** *A* path *is a graph P of the form*

$$V(P) = \{x_1, x_2, \ldots, x_l\}, \quad E(P) = \{x_1x_2, x_2x_3, \ldots, x_{l-1}x_l\}$$

This path P is usually denoted by $x_1x_2 \ldots x_l$ or $x_1 - x_2 - \ldots - x_l$. The vertices x_1 and x_l are the *endvertices* and $l - 1 = |E(P)|$ is the *length* of the path P. $\{x_1, \ldots x_{l-1}\}$ is the *inside* of the path P, denoted as P^*.

Graph G_0 is a path of length 3, with the inside $G_0^* = \{v_2, v_3\}$. If we would add any edge to G_0 it would stop being a path (sometimes we call such an edge a *chord*).

Definition 0.6 (connected graph, subset)**.** *A graph G is* connected *if for every pair $\{x, y\} \subseteq V(G)$ of distinct vertices, there is a path from x to y. A subset $X \subseteq V(G)$ is connected if the graph $G[X]$ is connected.*

formal enough?

Definition 0.7 (component)**.** *A* component *of a graph G is its maximal connected induced subgraph.*

Definition 0.8 (cycle)**.** *A* cycle *is a graph C of the form*

$$V(C) = \{x_1, x_2, \ldots, x_l\}, \quad E(C) = \{x_1x_2, x_2x_3, \ldots, x_{l-1}x_l, x_lx_1\}$$

This cycle C is usually denoted by $x_1x_2 \ldots x_lx_1$ or $x_1 - x_2 - \ldots - x_l - x_1$. $l = |E(C)|$ is the *length* of the cycle C. Sometimes we will denote the cycle of length l as C_l.

Notice, that a cycle is not a path (nor is a path a cycle). If we add an edge v_1v_4 to the path G_0 it becomes an even cycle C_4.

Definition 0.9 (hole)**.** *A* hole *is a cycle of length at least four.*

If a path, a cycle or a hole has an odd length, it will be called *odd*. Otherwise, it will be called *even*.

Definition 0.10 (complement)**.** *A* complement *of a graph $G = (V, E)$ is a graph $\overline{G} = (V, \binom{V}{2} \setminus E)$, that is two vertices x, y are adjacent in $\overline{G}$ iff they are not adjacent in G.*

Definition 0.11 (anticonnected graph, subset)**.** *A graph G is anticonnected if $\overline{G}$ is connected. A subset X is anticonnected if $\overline{G}[X]$ is connected.*

Definition 0.12 (anticomponent)**.** *An* anticomponent *of a graph G is an induced subgraph whose complement is a component in G.*

Definition 0.13 (clique)**.** *A* complete *graph or a* clique *is a graph of the form $G = (V, \binom{V}{2})$, that is every two vertices are connected. We will denote a clique of n vertices as K_n.*

Definition 0.14 (clique number)**.** *A* clique number *of a graph G, denoted as $\omega(G)$, is a cardinality of its largest induced clique.*

Definition 0.15 (anticlique)**.** *An* anticlique *is a graph in which there are no edges. We will also call anticliques* independent sets*.*

In a similar fashion, given a graph $G = (V, E)$, a subset of its vertices $V' \subseteq V$ will be called *independent* (in the context of G) iff $G[V']$ is an anticlique.

Definition 0.16 (stability number)**.** *A* stability number *of a graph G, denoted as $\alpha(G)$, is a cardinality of its largest induced stable set.*

Definition 0.17 (coloring)**.** *Given a graph G, its* coloring *is a function $c : V(G) \to \mathbb{N}^+$, such that for every edge $xy \in E(G)$, $c(x)$ is different from $c(y)$. A $k-$coloring of G (if exists) is a coloring, such that for all vertices $x \in V(G)$, $c(x) \leq k$.*

Definition 0.18 (chromatic number)**.** *A* chromatic number *of a graph G, denoted as $\chi(G)$, is a smallest natural number k, for which there exists a k-coloring of G.*

Definition 0.19 (Berge graph)**.** *A graph G is* Berge *if both G and $\overline{G}$ have no odd hole.*

Definition 0.20 (perfect graph)**.** *A graph G is* perfect *if for its every induced subgraph G' we have $\chi(G') = \omega(G')$*

Definition 0.21 (C-major vertices)**.** *Given a shortest off hole C in G, a node $v \in V(G) \setminus V(C)$ is $C-$major if the set of its neighbors in C is not contained in any 3-node path of C.*

a picture of this, clean odd hole, amenable hole

Definition 0.22 (clean odd hole)**.** *An odd hole C of G is* clean *if no vertex in G is C-major.*

Definition 0.23 (cleaner)**.** *Given a shortest odd hole C in G, a subset $X \subseteq V(G)$ is a* cleaner for C *if $X \cap V(C) = \emptyset$ and every C-major vertex belongs to X.*

Let us notice, that if X is a cleaner for C then C is a clean hole in $G \setminus X$.

Definition 0.24 (near-cleaner)**.** *Given a shortest odd hole C in G, a subset $X \subseteq V(G)$ is a* near-cleaner for C *if X contains all C-major vectices and $X \cap V(C)$ is a subset of vertex set of some 3-node path of C.*

Definition 0.25 (amenable odd hole)**.** *An odd hole C of G is* amenable *if it is a shortest off hole in G, it is of length at least 7 and for every anticonnected set X of C-major vectices there is a X-complete edge in C.*

Chapter 1

Perfect Graphs

Given a graph G, let us consider a problem of coloring it using as few colors as possible. If G contains a clique K as a subgraph, we must use at least $|V(K)|$ colors to color it. This gives us a lower bound for a chromatic number $\chi(G)$ – it is always greater or equal to the cardinality of the largest clique $\omega(G)$. The reverse is not always true, in fact we can construct a graph with no triangle and requiring arbitrarily large numbers of colors (e.g. construction by Mycielski [21]).

Do graphs that admit coloring using only $\omega(G)$ color are "simpler" to further analyze? Not necessarily so. Given a graph $G = (V, E)$, $|V| = n$, let us construct a graph G' equal to the union of G and a clique K_n. We can see that indeed $\chi(G') = n = \omega(G')$, but it gives us no indication of the structure of G or G'.

A definition of perfect graphs (definition 0.20) states that given a graph G, the chromatic number and cardinality of the largest clique of *its every induced subgraph* should be equal. The notion of perfect graphs was first introduced by Berge in 1961 [2] and it indeed captures some of the idea of graph being "simple" – in all perfect graphs the coloring problem, maximum (weighted) clique problem, and maximum (weighted) independent set problem can be solved in polynomial time [12]. Other classical NP-complete problems are still NP-complete in perfect graphs e.g. Hamiltonian path [20], maximum cut problem [3] or dominating set problem [10].

The most fundamental problem – the problem of recognizing perfect graphs – was open since its posing in 1961 until recently. Its solution, a polynomial algorithm recognizing perfect graphs is a union of the strong perfect graph theorem (section 1.1) stating that a graph is perfect if and only if it is Berge (definition 0.19) and an algorithm for recognizing Berge graphs in polynomial time (section 1.2).

all theese citations are about subclasses e.g. biparite graphs. Should we mention some subclasses?

Perfect graphs are interesting not only because of their theoretical properties, but they are also used in other areas of study e.g. integrality of polyhedra [8, 5], radio channel assignment problem [18, 19] and appear in the theory of Shannon capacity of a graph [14]. Also, as pointed out in [22, 5] algorithms to solve semi-definite programs grew out of the theory of perfect graphs. We will take a look

at semi-definite programs and at perfect graph's relation to Shannon capacity in section 1.3.1.

1.1 Strong Perfect Graph Theorem

The first step to solve the problem of recognizing perfect graphs was the *(weak) perfect graph theorem* first conjured by Berge in 1961 [2] and then proven by Lovász in 1972 [15].

Theorem 1.1.1 (Perfect graph theorem)**.** *A graph is perfect if and only if its complement graph is also perfect.*

Should we give some proof of that here? Maybe based on proof in [9]

Odd holes are not perfect, since their chromatic number is 3 and their largest cliques are of size 2. It is also easy to see, that an odd antihole of size n has a chromatic number of $\frac{n+1}{2}$ and largest cliques of size $\frac{n-1}{2}$. A graph with no odd hole and no odd antihole is called *Berge* (definition 0.19) after Claude Berge who studied perfect graphs.

In 1961 Berge conjured that a graph is perfect iff it contains no odd hole and no odd antihole in what has become known as a strong perfect graph conjecture. In 2001 Chudnovsky et al. have proven it and published the proof in an over 150 pages long paper "The strong perfect graph theorem" [7]. The following overview of the proof will be based on this paper and on an article with the same name by Cornuéjols [9].

Theorem 1.1.2 (Strong perfect graph theorem)**.** *A graph is perfect if and only if it is Berge.*

How long and detailed overview of the proof should we provide?

Znakomite pytanie. Generalnie ponieważ dowód jest drobiazgowy i trikowy, to wystarczy "z lotu ptaka" tj. to, co Chudnovsky i Seymour piszą w pracach popularnych i referują w wystąpieniach gościnnych typu te nagrania na YT. Głębiej nie ma sensu. Warto podkreślić, że technika dowodu i sam algorytm są zależne na dużo głębszym poziomie niż widać tj. ktoś mając dowód Stron Perfect Graph Theorem pewnie nie rozgryzłby algorytmu i vice versa.

1.2 Recognizing Berge Graphs

The following is based on the paper by Maria Chudnovsky et al. "Recognizing Berge Graphs". We will not provide full proof of its correctness, but will aim to show the intuition behind the algorithm.

Berge graph recognition algorithm could be divided into two parts: first we check if either G or $\overline{G}$ contain any of a number of simple structures as a induced subgraph (1.2.1). If they do, we output that graph is not Berge and stop. Else, we check if there is a near-cleaner for a shortest odd hole (1.2.2).

1.2.1 Simple structures

Pyramids

A *pyramid* in G is an induced subgraph formed by the union of a triangle [1] $\{b_1, b_2, b_3\}$, three paths $\{P_1, P_2, P_3\}$ and another vertex a, so that:

- $\forall_{1 \leq i \leq 3}$ P_i is a path between a and b_i
- $\forall_{1 \leq i < j \leq 3}$ a is the only vertex in both P_i and P_j and $b_i b_j$ is the only edge between $V(P_i) \setminus \{a\}$ and $V(P_j) \setminus \{a\}$.
- a is adjacent to at most one of $\{b_1, b_2, b_3\}$.

We will say that a can be *linked onto* the triangle $\{b_1, b_2, b_3\}$ *via* the paths P_1, P_2, P_3. Let us notice, that a pyramid is determined by its paths P_1, P_2, P_3.

It is easy to see that every graph containing a pyramid contains an odd hole – at least two of the paths P_1, P_2, P_3 will have the same parity.

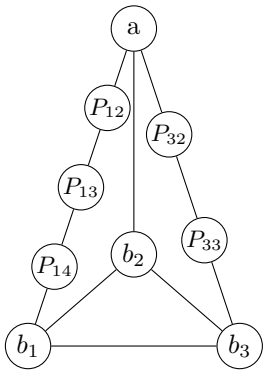


Figure 1.1: An example of a pyramid.

Finding Pyramids First, let us enumerate all 6-tuples $b_1, b_2, b_3, s_1, s_2, s_3$ such that:

- $\{b_1, b_2, b_3\}$ is a triangle
- for $1 \leq i < j \leq 3$, b_i, s_i is disjoint from b_j, s_j and $b_i b_j$ is the only edge between them
- there is a vertex a adjacent to all of s_1, s_2, s_3 and to at most one of b_1, b_2, b_3, such that if a is adjacent to b_i, then s_i b_i.

There are $O(|V(G)|^6)$ 6-tuples, and it takes $O(|V(G)|)$ time to check each one. For each such 6-tuple we follow with the rest of the algorithm.

We define $M = V(G) \setminus \{b_1, b_2, b_3, s_1, s_2, s_3\}$. Now, for each $m \in M$, we set $S_1(m)$ equal to the shortest path between s_1 and m such that s_2, s_3, b_2, b_3 have no neighbors in its interior, if such a path exists. We set S_2 and S_3 similarly. Then similarly we set $T_1(m)$ to be the shortest path between m and b_1, such that s_2, s_3, b_2, b_3 have no neighbors in its interior, if such a path exists. We do similar for T_2 and T_3. It takes $O(|V(G)|^2)$ time to calculate paths $T_i(m)$ for all i and m.

Now, we will calculate all possible paths P_i. For each $m \in M \cup \{b_1\}$ we will define a path $P_1(m)$ and paths $P_2(m)$, $P_3(m)$ will be defined in a similar manner.

[1] A triangle is a clique K_3.

If $s_1 = b_1$ let $P_1(b_1)$ be the one-vertex path with vertex b_1, and let $P_1(m)$ be undefined for each $m \in M$.

If $s_1 \neq b_1$, then $P_1(b_1)$ is undefined and for all $m \in M$ we will check if all the following are true:

- m is nonadjacent to all of b_2, b_3, s_2, s_3
- $S_1(m)$ and $T_1(m)$ both exist
- $V(S_1(m) \cap T_1(m)) = \{m\}$
- there are no edges between $V(S_1(m) \setminus m)$ and $V(T_1(m) \setminus m)$

If so, then we assign a path $s_1 - S_1(m) - m - T_1(m) - b_1$ to $P_1(m)$, otherwise we let $P_1(m)$ be undefined. It takes $O(|V(G)|^2)$ to check this, given m. We assign P_2 and P_3 in a similar manner. Total time of finding all $P_i(m)$ paths for a given 6-tuple is $O(|V(G)|^3)$.

Now we want to check if there is a triple m_1, m_2, m_3, so that $P_1(m_1)$, $P_2(m_2)$, $P_3(m_3)$ form a pyramid. A most obvious approach of enumerating them all would be too slow, so we do it carefully.

For $1 \leq i < j \leq 3$, we say that (m_i, m_j) is a *good* (i, j)*-pair*, iff $m_i \in M \cup \{b_i\}$, $m_j \in M \cup \{b_j\}$, $P_i(m_i)$, $P_j(m_j)$ both exist and the sets $V(P_i(m_i))$,$V(P_j(m_j))$ are both disjoint and $b_i b_j$ is the only edge between them.

We show how to find the list of all good $(1, 2)$-pairs, with similar algorithm for all other good (i, j)-pairs. For each $m_1 \in M \cup \{b_1\}$, we find the set of all m_2 such that (m_1, m_2) is a good $(1, 2)$-pair as follows.

If $P_1(m_1)$ does not exist, there are no such good pairs. If it exists, color black the vertices of M that either belong to $P_1(m_1)$ or have a neighbor in $P_1(m_1)$. Color all other vertices white. (We can do this in $O(|V(G)|^2)$) Then for each $m_2 \in M \cup \{b_2\}$, test whether $P_2(m_2)$ exists and contains no black vertices. We do this for all m_1 and get a set of all $(1, 2)$-good pairs. In similar way we calculate all good $(1, 3)$-pair and $(2, 3)$-pairs (in $O(|V(G)|^3)$ time).

Now, we examine all triples m_1, m_2, m_3 such that $m_i \in M \cup \{b_i\}$ and test whether (m_i, m_j) is a good (i, j)-pair. If we find a triple such that all three pairs are good, we output that G contains a pyramid and stop.

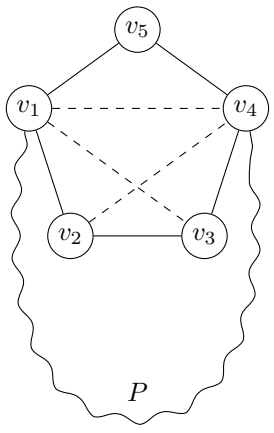
If after examining all choices of $b_1, b_2, b_3, s_1, s_2, s_3$ we find no pyramid, output that G contains no pyramid. Since there are $O(|V(G)|^6)$ such choices and it takes a time of $O(|V(G)|^3)$ to analyze each one, the total time is $O(|V(G)|^9)$.

some proofs

Jewels

Five vertices $v_1, \ldots, v_5$ and a path P form a *jewel* iff:

- $v_1, \ldots, v_5$ are distinct vertices.
- $v_1v_2, v_2v_3, v_3v_4, v_4v_5, v_5v_1$ are edges.



- v_1v_3, v_2v_4, v_1, v_4 are nonedges.
- P is a path between v_1 and v_4, such that v_2, v_3, v_5 have no neighbors in its inside.

Most obvious way to find a jewel would be to enumerate all choices of $v_1, \dots v_5$, check if a choice is correct and if it is try to find a path P as required. This gives us a time of $O(|V|^7)$. We could speed it up to $O(|V|^6)$ with more careful algorithm, but since whole algorithms takes time $O(|V|^9)$ and our testing showed that time it takes to test for jewels is negligible we decided against it.

Configurations of type $\mathcal{T}_1$

A configuration of type $\mathcal{T}_1$ is a hole of length 5. To find it we simply iterate all choices of paths of length of 4, and check if there exists a fifth vertex to complete the hole. See section 2.1.1 for more implementation details.

Configurations of type $\mathcal{T}_2$

A configuration of type $\mathcal{T}_2$ is a tuple $(v_1, v_2, v_3, v_4, P, X)$, such that:

- $v_1v_2v_3v_4$ is a path in G.
- X is an anticomponent of the set of all $\{v_1, v_2, v_4\}$-complete vertices.
- P is a path in $G \setminus (X \cup \{v_2, v_3\})$ between v_1 and v_4 and no vertex in P^* is X-complete or adjacent to v_2 or adjacent to v_3.

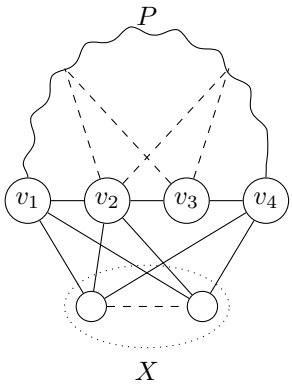


Figure 1.3: An example of a $\mathcal{T}_2$.

Checking if configuration of type $\mathcal{T}_2$ exists in our graph is straightforward: we enumerate all paths $v_1 \dots v_4$, calculate set of all $\{v_1, v_2, v_4\}$-complete vertices and its anticomponents. Then, for each anticomponent X we check if required path P exists.

To prove that existence of $\mathcal{T}_2$ configuration implies that the graph is not berge, we will need the following Roussel-Rubio lemma:

Lemma 1.2.1 (Roussel-Rubio Lemma [23, 6])**.** *Let G be Berge, X be an anticonnected subset of $V(G)$, P be an odd path $p_1 \dots p_n$ in $G \setminus X$ with length at least 3, such that p_1 and p_n are X-complete and $p_2, \dots, p_{n-1}$ are not. Then:*

- *P is of length at least 5 and there exist nonadjacent $x, y \in X$, such that there are exactly two edges between x, y and P^*, namely xp_2 and yp_{n-1},*
- *or P is of length 3 and there is an odd antipath joining internal vertices of P with interior in X.*

We may use this lemma quite often, might want to provide proof if so.

Now, we shall prove the following:

Lemma 1.2.2. *If G contains configuration of type $\mathcal{T}_2$ then G is not Berge.*

Proof. Let $(v_1, v_2, v_3, v_4, P, X)$ be a configuration of type $\mathcal{T}_2$. Let us assume that G is not Berge and consider the following:

- If P is even, then $v_1, v_2, v_3, v_4, P, v_1$ is an odd hole,
- If P is of length 3. Let us name its vertices v_1, p_2, p_3, v_4. It follows from Lemma 1.2.1, that there exists an odd antipath between p_2 and p_3 with interior in X. We can complete it with v_2p_2 and v_2p_3 into an odd antihole.
- If P is odd with the length of at least 5 , it follows from Lemma 1.2.1 that we have $x, y \in X$ with only two edges to P being xp_2 and yp_{n-1}. This gives us an odd hole: $v_2, x, p_2, \ldots, p_{n-1}, y, v_2$.

□

I merged a couple of proofs from [7], check in the morning if this is correct.

check in the morning

Configurations of type $\mathcal{T}_3$

A configuration of type $\mathcal{T}_3$ is a sequence $v_1, \ldots, v_6$, P, X, such that:

- $v_1, \ldots v_6$ are distinct vertices.
- v_1v_2, v_3v_4, v_1v_4, v_2v_3, v_3v_5, v_4v_6 are edges, and v_1v_3, v_2v_4, v_1v_5, v_2v_5, v_1v_6, v_2v_6, v_4v_5 are nonedges.
- X is an anticomponent of the set of all $\{v_1, v_2, v_5\}$-complete vertices, and v_3, v_4 are not X-complete.
- P is a path of $G \setminus (X \cup \{v_1, v_2, v_3, v_4\})$ between v_5 and v_6 and no vertex in $P*$ is X-complete or adjacent to v_1 or adjacent to v_2.
- If v_5v_6 is an edge, then v_6 is not X-complete.

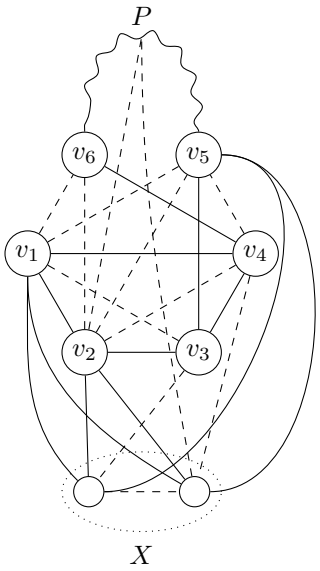


Figure 1.4: An example of a $\mathcal{T}_3$.

The following algorithm with running time of $O(|V(G)|^6)$ checks whether G contains a configuration of type T_3:

For each triple v_1, v_2, v_5 of vertices such that v_1v_2 is an edge and v_1v_5, v_2v_5 are nonedges find the set Y of all $\{v_1, v_2, v_5\}$-complete vertices. For each anticomponent X of Y find the maximal connected subset F' containing v_5 such that v_1, v_2 have no neighbors in F' and no vertex of $F' \setminus \{v_5\}$ is X-complete. Let F be the union of F' and the set of all X-complete vertices that have a neighbor in F' and are nonadjacent to all of v_1, v_2 and v_5.

Then, for each choice of v_4 that is adjacent to v_1 and not to v_2 and v_5 and has a neighbor in F (call it v_6) and a nonneibhbor in X, we test whether there is a vertex v_3, adjacent to v_2, v_4, v_5 and not to v_1, with a nonneibhbor in X. If there is such a vertex v_3, find P – a path from v_6 to v_5 with interior in F' and return that $v_1, \dots v_6, P, X$ is a configuration of type $\mathcal{T}_3$. If we exhaust our search and find none, report that graph does not contain it.

To see that the algorithm below has a running time of $O(|V(G)|^6)$, let us note that for each triple v_1, v_2, v_5 we examine, of which there are $O(|V(G)|^3)$, there are linear many choices of X, each taking $O(|V(G)|^2)$ time to process and generating a linear many choices of v_4 which take a linear time to process in turn. This gives us the total running time of $O(|V(G)|^6)$.

We will skip the proof that each graph containing $\mathcal{T}_3$ is not Berge. See section6 6.7 of [6] for the proof.

1.2.2 Amenable holes.

Theorem 1.2.3. *Let G be a graph, such that G and $\overline{G}$ contain no Pyramid, no Jewel and no configuration of types $\mathcal{T}_1, \mathcal{T}_2$ or $\mathcal{T}_3$. Then every shortest hole in G is amenable.*

The proof of this theorem is quite technical and we will not discuss it here. See section 8 of [6] for the proof.

With theorem 1.2.3 we can describe the rest of the algorithm.

Algorithm 1.2.1 (List possible near cleaners, 9.2 of [6])
Input: A graph G.
Output: $O(|V(G)|^5)$ subsets of $V(G)$, such that if C is an amenable hole in G, then one of the subsets is a near-cleaner for C.

Begin.
We will call a triple (a, b, c) of vertices *relevant* if a, b are distinct (possibly $c \in \{a, b\}$) and $G[\{a, b, c\}]$ is an independent set.

Given a relevant triple (a, b, c) we can compute the following:

- $r(a, b, c) \leftarrow$ the cardinality of the largest anticomponent of $N(a, b)$, that contains a nonneibhbor of c, or 0, if c is $N(a, b)$-complete.

- $Y(a,b,c) \leftarrow$ the union of all anticomponents of $N(a,b)$ that have cardinality strictly greater than $r(a,b,c)$.
- $W(a,b,c) \leftarrow$ the anticomponent of $N(a,b) \cup \{c\}$ that contains c.
- $Z(a,b,c) \leftarrow$ the set of all $Y(a,b,c) \cup W(a,b,c)$-complete vertices.
- $X(a,b,c) \leftarrow Y(a,b,c) \cup Z(a,b,c)$.

For every two adjacent vertices u, v compute the set $N(u,v)$ and list all such sets. For each relevant triple (a,b,c) compute the set $X(a,b,c)$ and list all such sets.

Output all subsets of $V(G)$ that are the union of a set from the first list and a set from the second list. *End*

To proove the correctness of algorithm 1.2.1 we will need the following theorem.

Theorem 1.2.4 (9.1 of [6])**.** *Let C be a shortest odd hole in G, with length at least 7. Then there is a relevant triple (a,b,c) of vertices such that*

- *the set of all C-major vertices not in $X(a,b,c)$ is anticonnected*
- *$X(a,b,c) \cap V(C)$ is a subset of the vertex set of some 3-vertex path of C.*

Proof.
do we want it here? 1-2 pages long

□

Let us suppose that C is an amenable hole in G. By theorem 1.2.4, there is a relevant triple (a,b,c) satisfying that theorem. Let T be the set of all C-major vertices not in $X(a,b,c)$. From theorem 1.2.4 we get that T is anticonnected. Since C is amenable, there is an edge uv of C that is T-complete, and therefore $T \subseteq N(u,v)$. But then $N(u,v) \cup X(a,b,c)$ is a near-cleaner for C. Therefore the output of the algorithm 1.2.1 is correct.

Algorithm 1.2.2 (Test possible near clener, 5.1 of [6])
Input: A graph G containing no simple bad structure , and a subset $X \subseteq V(G)$. define this somewhere
Output: Determines one of the following:

- *G has an odd hole*
- *There is no shortest odd hole C such that X is a near-cleaner for C.*

Begin.
For every pair $x, y \in V(G)$ of distinct vertices find shortest path $R(x,y)$ between x, y with no internal vertex in X. If there is one, let $r(x,y)$ be its length, if not, let $r(x,y)$ be infinite.

For all $y_1 \in V(G) \setminus X$ and all 3-vertex paths $x_1 - x_2 - x_3$ of $G \setminus y_1$ we check the following:

- $R(x_1, y_1), R(x_2, y_2)$ both exist – define y_2 as the neighbor of y_1 in $R(x_2, y_1)$.
- $r(x_2, y_1) = r(x_1, y_1) + 1 = r(x_1, y_2)$ $(= n$ say$)$
- $r(x_3, y_1), r(x_3, y_2) \geq n$

If there is such a choice of x_1, x_2, x_3, y_1 then we output that there is an odd hole. If not, we report that there is no shortest odd hole C such that X is a near-cleaner for C. *End*

Below we will proove that if the algorithm 1.2.2 reports an odd hole in G, there indeed is one. The proof of the correctness of the other possible result is more complicated, see section 4 and theorem 5.1 of [6].

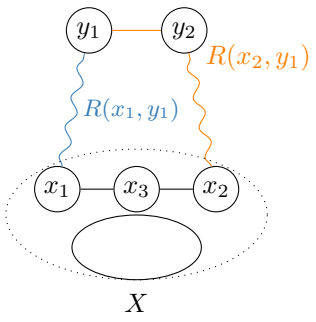


Figure 1.5: An odd hole is found

Proof. Let us suppose that there is a choice of x_1, x_2, x_3, y_1 satisfying the three conditions in the algorithm 1.2.2 and let y_2 and n be defined as in there. We claim that G contains an odd hole.

Let $R(x_1, y_1) = p_1 - \ldots - p_n$, and let $R(x_2, y_1) = q_1 - \ldots - q_{n+1}$, where $p_1 = x_1$, $p_n = q_{n+1} = y_1$, $q_1 = x_2$ and $q_n = y_2$. From the definition of $R(x_1, y_1)$ and $R(x_2, y_1)$, none of $p_2, \ldots, p_{n-1}, q_2, \ldots, q_n$ belong to X. Also, from the definition of y_1, $y_1 \notin X$.

Since $r(x_1, y_1) = r(x_2, y_1) - 1$ and since x_1, x_2 are nonadjacent it follows that x_2 does not belong to $R(x_1, y_1)$ and x_1 does not belong to $R(x_2, y_1)$. Since $r(x_3, y_1), r(x_3, y_2) \geq n (= r(x_1, y_2))$ it follows that x_3 does not belong to $R(x_1, y_1)$ or to $R(x_2, y_1)$, and has no neighbors in $R(x_1, y_1) \cup R(x_2, y_1)$ other than x_1, x_2. Since $r(x_1, y_2) = n$ we get that y_2 does not belong to $R(x_1, y_1)$.

We claim first that the insides of paths $R(x_1, y_1)$ and $R(x_2, y_1)$ have no common vertices. For suppose that there are $2 \leq i \leq n-1$ and $2 \leq j \leq n$ that $p_i = q_j$. Then the subpaths of these two paths between p_i, y_1 are both subpaths of the shortest paths, and therefore have the same length, that is $j = i + 1$. So $p_1 - \ldots - p_1 - q_{j+1} - \ldots - q_n$ contains a path between x_1, y_2 of length $\leq n - 2$, contradicting that $r(x_1, y_2) = n$. So $R(x_1, y_1)$ and $R(x_2, y_1)$ have no common vertex except y_1.

If there are no edges between $R(x_1, y_1) \setminus y_1$ and $R(x_2, y_1) \setminus y_1$ then the union of these two paths and a path $x_1 - x_3 - x_2$ form an odd hole, so the answer is correct.

Suppose that $p_i q_j$ is an edge for some $1 \leq i \leq n-1$ and $1 \leq j \leq n$. We claim $i \geq j$. If $j = 1$ this is clear so let us assume $j > 1$. Then there is a path between x_1, y_2 within $\{p_1, \ldots, p_i, q_j, \ldots q_n\}$ which has length $\leq n - j + 1$ and has no internal vertex in X (since $j > 1$); and since $r(x_1, y_2) = n$, it follows that $n - j + i \geq n$, that is, $i \geq j$ as claimed.

Now, since x_1, x_2 are nonadjacent $i \geq 2$. But also $r(x_2, y_1) \geq n$ and so $j + n - i \geq n$, that is $j \geq i$. So we get $i = j$. Let us choose i minumum,

then $x_3 - x_1 - \ldots - p_i - q_i - \ldots - x_2 - x_3$ is an odd hole, which was what we wanted. □

1.3 Coloring Perfect Graphs

A natural problem for perfect graphs is a problem of coloring them. In 1988 Martin Grötschel et al. published an ellipsoid-method-based polynomial algorithm for coloring perfect graphs [11]. We consider it in section section 1.3.1. Hovewer due to its use of the ellipsoid method this algorithm has been usually considered unpractical [4, 5, 13].

There has been much progress on the quest of finding a more classical algorithm coloring perfect graphs, without the use of ellipsoid method (see section 1.3.4), however there is still no known polynomial combinatorial algorithm to do this.

better wording of this paragraph

1.3.1 Information theory background

Shannon Capacity of a graph

The polynomial technique of coloring perfect graphs known so far arose in the field of semidefinite programming. Semidefinite programs are linear programs over the cones of semi-definite matrice. The connection of coloring graphs and the cones of semi-definite matrices might be surprising, so let us take a brief digression into the field of information theory, where we will see the connection more clearly. Also, this is the background which motivated Berge to introduce perfect graphs [5].

Suppose we have a noisy communication channel in which certain signal values can be confused with others. For instance, suppose our channel has five discrete signal values, represented as 0, 1, 2, 3, 4. However, each value of i when sent across the channel can be confused with value $(i \pm 1)$ mod 5. This situation can be modeled by a graph C_5 (section 1.3.1) with in which vertices correspond to signal values and two vertices are connected iff values they represent can be confused.

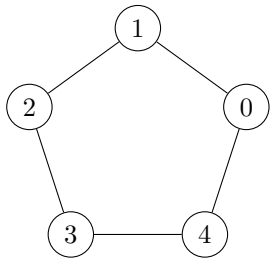


Figure 1.6: Example noisy channel

We are interested in transmission without possibility of confusion. For this example it is possible for two values to be transmited without ambiguity e.g. values 1 and 4, which allows us to send 2^n non-confoundable messages in n steps. But we could do better, for example we could communicate five two-step codewords e.g. ”00”, ”12”, ”24”, ”43”, ”31”. Each pair of these codewords includes at least one position where its values differ by two or more modulo 5, which allows the recipent to distinguish them without confusion. This allows us to send $5^{n/2}$ non-confoundable messages in n steps.

Let us be more precise. Given a graph G modeling a communication channel and a number $k \geq 1$ we say that two messages $v_1v_2 \dots v_k$, $w_1w_2 \dots w_k \in G(V)^k$ of lenght k are non-confoundable iff there is $1 \leq i \leq k$ such that v_i, w_i are non-confoundable. We are interested in the maximum rate at which we can reliably transmit information (the *Shannon capacity* of the channel defined by G) For $k = 1$, maximum number of messages we can send without confusion in a single step is equal to $\alpha(G)$.

To describe longer messages we indroduce *Strong Product* $G \cdot H$ of two graphs $G = (V, E)$, $H = (W, F)$ as the graph with $V(G \cdot H) = V \times W$, with $(i, u)(j, v) \in E(G \cdot H)$ iff $ij \in E$ and $uv \in F$, or $ij \in E$ and $u = v$, or $i = j$ and $uv \in F$. Given channel modeled by G it is easy to see that the maximum number of distinguishable words of length 2 is equal to $\alpha(G \cdot G)$, and in general the number of distinguishable words of length k is equal to $\alpha(G^k)$ – which gives us $\sqrt[k]{\alpha(G^k)}$ as the number of distinguishable signals per single transmission. So, we can define the Shannon capacity of the channel defined by G as $\Theta(G) = \sup_k \sqrt[k]{\alpha(G^k)}$.

Unfortunately, it is not known whether $\Theta(G)$ can be computed for all graphs (polynomial or not). If we could calculate $\alpha(G^k)$ for a first few values of k (we will show how to do it in algorithm 1.3.1) we could have a lower bound on $\Theta(G)$. Let us now turn into search for some usable upper bound.

Lovász number

The following introduction of what has become known as *Lovász number* is based on excellent lecture notes by Lovász [16].

rewrite this so its not blatantly copied.

For a channel defined by graph C_5 using five messages of length 2 gives us a lower bound on $\Theta(G)$ equal $\sqrt{5}$ (as does calculating $\alpha(C_5^2)$).

Consider an "umbrella" in $\mathbb{R}^3$ with the unit vector $e_1 = (1, 0, 0)$ as its "handle" and 5 "ribs" of unit length . Open it up to the point where non-consecutive ribs are orthogonal, that is form an angle of 90°. This way we get a representation of C_5 by 5 unit vectors $u_1, \dots u_5$ so that each u_i forms the same angle with e_1 and any two non-adjacent nodes are represented with orthogonal vectors. We can calculate $e_1^\intercal u_i = 5^{-1/4}$.

picture

It turns out, that we can obtain a similar representation of the nodes of C_5^k by unit vectors $v_i \in \mathbb{R}^3k$, so that any two non-adjacent nodes are labeled with orthogonal vectors (this representation is sometimes called the *orthogonal representation* [17]). Moreover, we still get $e_1^\intercal v_i = 5^{-k/4}$ for every $i \in V(C_5^k)$ (the proof is quite technical and we omit it here).

If S is any stable set in C_5^k, then $\{v_i, i \in S\}$ is a set of mutually orthogonal unit vectors so we get

$$\sum_{i \in S} (e_1^\intercal v_i)^2 \leq |e_1|^2 = 1$$

(if v_i formed a basis then this inequality would be an equality).

On the other hand each term on the left hand side is $5^{-1/4}$, so the left hand side is equal to $|S|5^{-k/2}$, and so $|S| \leq 5^{k/2}$. Thus we get $\alpha(C_5^k) \leq 5^{k/2}$ and

$\Theta(C_5) = \sqrt{5}$.

This method extends to any graph G in place of C_5. All we have to do is find a orthogonal representation that will give us the best bound. So, we can define the *Lovász number* of a graph G as :

$$\vartheta(G) = \min_{c,U} \max_{i \in V} \frac{1}{(c^\intercal u_i)^2},$$

this equation does not really follow from the thought process above, it is a slightly different definition

where c is a unit vector in $\mathbb{R}^{|V(G)|}$ and U is a orthogonal representation of G. But how can we construct an optimum (or even good) orthogonal representation? It turns out that it can be computed in polynomial time using semidefinite optimization.

1.3.2 Computing ϑ

TODO

1.3.3 Coloring perfect graph using ellipsoid method

Theorem 1.3.1 (Lovász "sandwich theorem"). *For any graph G:*

$$\omega(G) \leq \vartheta(\overline{G}) \leq \chi(G)$$

Because in perfect graphs $\omega(G) = \chi(G)$, we get $\omega(G) = \vartheta(G) = \chi(G)$.

TODO

Maximum cardinality stable set

cite the paper on using theta to color.

Given graph G, recall that stability number of G is equal clique number of the complement of G. This gives us a way to compute $\alpha(G)$ for any perfect graph G.

In fact, to calculate $\chi(\overline{G})$ and $\alpha(G)$ we only need an approximated value of $\vartheta(G)$ with precision $< \frac{1}{2}$.

We will now show how to find a stable set in G of size $\alpha(G)$.

Algorithm 1.3.1 (maximum cardinality stable set in a perfect graph)
Input: A perfect graph $G = (V, E)$.
Output: A maximum cardinality stable set in G.

Begin.
Let $v_1, \ldots v_n$ be an ordering of vertices of G. We will construct a sequence of induced subgraphs $G = G_0 \supseteq G_1 \supseteq \ldots \supseteq G_n$, so that G_n is a required stable set.

Let $G_0 \leftarrow G$. Then, for each $i \geq 1$, compute $\alpha(G_{i-1} \setminus v_i)$. If $\alpha(G_{i-1} \setminus v_i) = \alpha(G)$, then set $G_i \leftarrow G_{i-1} \setminus \{v_i\}$, else set $G_i \leftarrow G_{i-1}$.

Return G_n. *End*

Let us proove that G_n is indeed a stable set. Suppose otherwise and let v_iv_j be an edge in G_n with $i < j$ and i minimal. But then $\alpha(G_{i-1} \setminus v_i) = \alpha(G_{i-1}) = \alpha(G)$ so by our construction v_i is not in G_i and v_iv_j is not an edge of G_n. Therefore there are no edges in G_n.

Because at every step we have $\alpha(G_i) = \alpha(G_{i-1})$, therefore $\alpha(G_n) = \alpha(G)$, so G_n is required maximum cardinality stable set.

The running time of algorithm 1.3.1 is polynomial, because we construct n auxiliary graphs, each requiring calculating α once plus additional $O(|V|^2)$ time for constructing the graph.

Given a weight function $w : V \to \mathbb{N}$ we could calculate the maximum weighted stable set in G in the following manner. Create graph G' by replacing every node v by a set W_v of $w(v)$ nonadjacent nodes, making two nodes $x \in W_v$, $y \in W_u$ adjacent in G' iff the nodes v, u are adjacent in G. Then calculate a maximum cardinality stable set in G' (we remark that G' is still perfect because every new introduced hole is even) and return a result of those vertices in G whose any (and therefore all) copies were chosen. We will use this technique later on.

Stable set intersecting all maximum cardinality cliques

Next, let us show how to find a stable set intersecting all the maximum cardinality cliques of G.

Algorithm 1.3.2
Input: A perfect graph $G = (V, E)$.
Output: A stable set which intersects all the maximum cardinality cliques of G.

Begin.
We will create a list $Q_1, \ldots Q_t$ of all maximum cardinality cliques of G.

Let $Q_1 \leftarrow$ a maximum cardinality clique of G. We calculate this by running algorithm 1.3.1 on $\overline{G}$.

Now suppose $Q_1, \ldots, Q_t$ have been found. We show how to calculate Q_{t+1} or see that we are done.

Let us define a weight function $w : V \to \mathbb{N}$, so that for $v \in V$, $w(v)$ is equal to the number of cliques $Q_1, \ldots Q_t$ that contain v.

Assign $S \leftarrow$ the maximum w-weighted stable set, as described in a remark to algorithm 1.3.1. It is easy to see that S has weight t, which means that S meets each of $Q_1, \ldots Q_t$.

If $\omega(G \setminus S) < \omega(G)$, then S meets all the maximum cardinality cliques in G so we return S. Otherwise we find a maximum cardinality clique in $G \setminus S$ (it will be of size $\omega(G)$, because $\omega(G \setminus S) = \omega(G)$), add it to our list as Q_{t+1} and continue with longer list.

End

There are at most $|V|$ maximum cardinality cliques in G. Adding a single clique to the list of maximum cardinality cliques requires constructing auxiliary graph for weighted maximum stable set, which is of size $O(V^2)$ and running algorithm 1.3.1 on it. Therefore total running time is polynomial.

Minimum coloring

Algorithm 1.3.3
Input: A perfect graph $G = (V, E)$.
Output: A coloring of G using $\chi(G)$ colors.

Begin.
If G is equal to its maximum cardinality stable set, color all verces one color and return.

Else find S intersecting all maximum cardinality cliques of G (algorithm 1.3.2). Color recursively all vertices of $G \setminus S$ with $\chi(G \setminus S) = \omega(G \setminus S) = \omega(G) - 1$ colors and all vertices of S with one additional color. *End*

We will call recursion at most $O(|V|)$ times, each step of recursion is polynomial in time. Therefore the running time of algorithm 1.3.3 is polynomial.

1.3.4 Classical algorithms

Chapter 2

Implementation

2.1 Berge graphs recognition

Anything interesting about algo/data structure?

2.1.1 Optimizations

Bottlenecks in performance (next path, are vectors distinct etc).

In our graph preprocessing we have a pointer to next edge in order to speed up generating next path.

We used callgrind to get idea of methods crucial for time.

In general enumerating all paths is crucial. As is checking if vector has distinct values.

Jewels – we iterate all possibly chordal paths and check if they are ok - much faster

$\mathcal{T}_1$ – we iterate all paths of length 4 and check if there exists a fifth vertex to complete the hole - much faster than iterating vertices.

Validity tests - unit tests, tests of bigger parts, testing vs known answer and vs naive.

2.1.2 Parallelism with CUDA (?)

TODO

2.1.3 Experiments

Naive algorithm - brief description, bottlenecks optimizations (makes huge difference).

Description of tests used.

Results and Corollary - almost usable algorithm.

2.2 Coloring Berge Graphs

2.2.1 Ellipsoid method

Description.

Implementation.

Experiments and results.

2.2.2 Combinatorial Method

Cite the paper.

On its complexity - point to appendix for pseudo-code.

Appendices

Appendix A

Perfect Graph Coloring algorithm

TODO

Bibliography

[1] Béla Bollobás. *Modern graph theory*. New York: Springer, 1998. ISBN: 978-0-387-98488-9.

[2] C. Berge. "Färbung von Graphen, deren sämtliche beziehungsweise deren ungerade Kreise starr sind". In: Wissenschaftliche Zeitschrift der Martin-Luther-Universität Halle-Wittenberg, Mathematisch-naturwissenschaftliche Reihe, 1961, p. 114.

[3] Hans L. Bodlaender and Klaus Jansen. "On the complexity of the maximum cut problem". In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1994, pp. 769–780. DOI: 10.1007/3-540-57785-8_189. URL: https://doi.org/10.1007/3-540-57785-8_189.

[4] Maria Chudnovsky et al. "Coloring square-free Berge graphs". In: *Journal of Combinatorial Theory, Series B* 135 (Mar. 2019), pp. 96–128. DOI: 10.1016/j.jctb.2018.07.010. URL: https://doi.org/10.1016/j.jctb.2018.07.010.

[5] Maria Chudnovsky et al. "Progress on perfect graphs". In: *Mathematical Programming* 97.1 (July 2003), pp. 405–422. DOI: 10.1007/s10107-003-0449-8. URL: https://doi.org/10.1007/s10107-003-0449-8.

[6] Maria Chudnovsky et al. "Recognizing Berge Graphs". In: *Combinatorica* 25.2 (Mar. 2005), pp. 143–186. DOI: 10.1007/s00493-005-0012-8. URL: https://doi.org/10.1007/s00493-005-0012-8.

[7] Maria Chudnovsky et al. "The strong perfect graph theorem". In: *Annals of Mathematics* 164.1 (July 2006), pp. 51–229. DOI: 10.4007/annals.2006.164.51. URL: https://doi.org/10.4007/annals.2006.164.51.

[8] V Chvátal. "On certain polytopes associated with graphs". In: *Journal of Combinatorial Theory, Series B* 18.2 (Apr. 1975), pp. 138–154. DOI: 10.1016/0095-8956(75)90041-6. URL: https://doi.org/10.1016/0095-8956(75)90041-6.

[9] G. Cornuéjols. "The strong perfect graph theorem". In: *Optima* 70 (June 2003), pp. 2–6. DOI: 10.4007/annals.2006.164.51. URL: https://www.andrew.cmu.edu/user/gc0v/webpub/optima.pdf.

[10] A. K. Dewdney. “Reductions Between NP-complete Problems”. In: *Fast Turing Reductions Between Problems in NP*. University of Western Ontario: Department of Computer Science, University of Western Ontario, 1981. Chap. 4. ISBN: 9780771403057. URL: https://doi.org/10.1007/978-3-642-78240-4.

[11] Martin Grötschel, László Lovász, and Alexander Schrijver. “Coloring Perfect Graphs”. In: *Geometric Algorithms and Combinatorial Optimization*. Springer Berlin Heidelberg, 1993. Chap. 9.1, pp. 296–299. DOI: 10.1007/978-3-642-78240-4. URL: https://doi.org/10.1007/978-3-642-78240-4.

[12] Martin Grötschel, László Lovász, and Alexander Schrijver. “Stable Sets in Graphs”. In: *Geometric Algorithms and Combinatorial Optimization*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993. Chap. 9, pp. 273–303. ISBN: 9783642782404. URL: https://doi.org/10.1007/978-3-642-78240-4.

[13] Benjamin Lévêque et al. “Coloring Artemis graphs”. In: *Theoretical Computer Science* 410.21-23 (May 2009), pp. 2234–2240. DOI: 10.1016/j.tcs.2009.02.012. URL: https://doi.org/10.1016/j.tcs.2009.02.012.

[14] L. Lovasz. “On the Shannon capacity of a graph”. In: *IEEE Transactions on Information Theory* 25.1 (Jan. 1979), pp. 1–7. DOI: 10.1109/tit.1979.1055985. URL: https://doi.org/10.1109/tit.1979.1055985.

[15] L. Lovász. “Normal hypergraphs and the perfect graph conjecture”. In: *Discrete Mathematics* 2.3 (June 1972), pp. 253–267. DOI: 10.1016/0012-365x(72)90006-4. URL: https://doi.org/10.1016/0012-365x(72)90006-4.

[16] L. Lovász. *Semidefinite programs and combinatorial optimization (Lecture notes)*. 1995. DOI: https://doi.org/10.1007/0-387-22444-0_6. URL: http://www.cs.elte.hu/~lovasz/semidef.ps.

[17] L. Lovász, M. Saks, and A. Schrijver. “Orthogonal representations and connectivity of graphs”. In: *Linear Algebra and its Applications* 114-115 (Mar. 1989), pp. 439–454. DOI: 10.1016/0024-3795(89)90475-8. URL: https://doi.org/10.1016/0024-3795(89)90475-8.

[18] Colin McDiarmid. *Graph imperfection and channel assignment*. 1999. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.98.9490.

[19] Colin McDiarmid and Bruce Reed. “Channel assignment and weighted coloring”. In: *Networks* 36.2 (2000), pp. 114–117. DOI: 10.1002/1097-0037(200009)36:2<114::aid-net6>3.0.co;2-g. URL: https://doi.org/10.1002/1097-0037(200009)36:2%3C114::aid-net6%3E3.0.co;2-g.

[20] Haiko Müller. "Hamiltonian circuits in chordal bipartite graphs". In: *Discrete Mathematics* 156.1-3 (Sept. 1996), pp. 291–298. DOI: 10.1016/0012-365x(95)00057-4. URL: https://doi.org/10.1016/0012-365x(95)00057-4.

[21] Jan Mycielski. "Sur le coloriage des graphs". In: *Colloquium Mathematicum* 3.2 (1955), pp. 161–162. DOI: 10.4064/cm-3-2-161-162. URL: https://doi.org/10.4064/cm-3-2-161-162.

[22] Jorge L. Ramírez Alfonsin and Bruce A. Reed, eds. *Perfect graphs*. Wiley-Interscience series in discrete mathematics and optimization. Chichester ; New York: Wiley, 2001. ISBN: 9780471489702.

[23] F. Roussel and P. Rubio. "About Skew Partitions in Minimal Imperfect Graphs". In: *Journal of Combinatorial Theory, Series B* 83.2 (Nov. 2001), pp. 171–190. DOI: 10.1006/jctb.2001.2044. URL: https://doi.org/10.1006/jctb.2001.2044.