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

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.

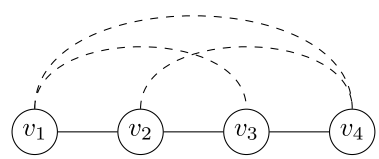

Figure 1: An example graph G_0

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*

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

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

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

This cycle C is usually denoted by $x_1x_2 \dots x_lx_1$ or $x_1 - x_2 - \dots - x_l - x_1$. $l = |E(C)|$ 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 (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 0.15 (clique number)**.** *A* clique number *of a graph G, denoted as $\omega(G)$, is a cardinality of its largest induced clique.*

Definition 0.16 (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 0.17 (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 0.18 (Berge graph)**.** *A graph G is* Berge *if both G and $\overline{G}$ have no odd hole.*

Definition 0.19 (perfect graph)**.** *A graph G is* perfect *if for its every induced subgraph G' we have $\chi(G') = \omega(G')$*

Definition 0.20 (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 0.21 (clean odd hole)**.** *An odd hole C of G is* clean *if no vertex in G is C-major.*

Definition 0.22 (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 0.23 (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 0.24 (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 [8]).

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 0.19) 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 the idea of graph being "simple" – in all perfect graphs the coloring problem, maximum clique problem, and maximum independent set problem can be solved in polynomial time [6].

One of properties of perfect graphs is the *perfect graph theorem* first conjured by Berge in 1961 [2] and then proven by Lovász in 1972 [7].

Theorem 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 [5]

1.1 Strong Perfect Graph Theorem

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 0.18) 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" [4]. The following overview of the proof will be based on this paper and on an article with the same name by Cornuéjols [5].

Theorem 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.

Chapter 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.

2.1 Recognition algorithm Overview

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 (2.1.1). 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.(2.1.2).

2.1.1 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.

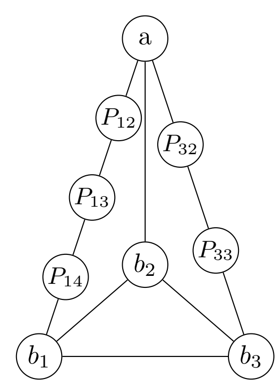

Figure 2.1: An example of a pyramid.

[1] A triangle is a clique K_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.

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.

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.

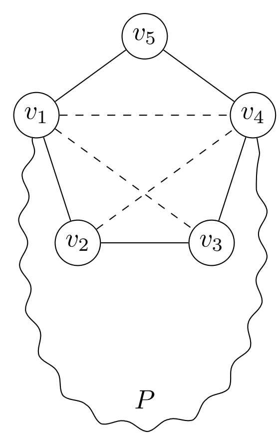

Figure 2.2: An example of a jewel.

Most obvious way to find a jewel would be to enumerate all choices of $v_1, \ldots v_5$, 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 2.2.1 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.

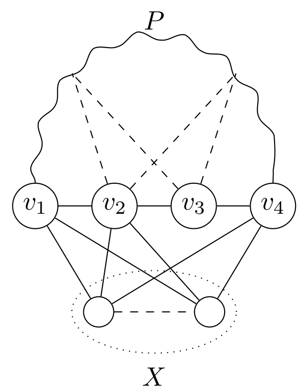

Figure 2.3: An example of a $\mathcal{T}_2$.

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 2.1 (Roussel-Rubio Lemma [9, 3])**.** *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 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 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 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: $v_2, x, p_2, \dots, p_{n-1}, y, v_2$.

I merged a couple of proofs from [4], check in the morning if this is correct.

check in the morning

$\square$

Configurations of type $\mathcal{T}_3$

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

- $v_1, \dots v_6$ 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.

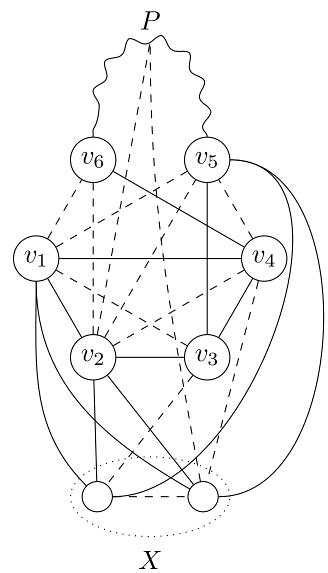

Figure 2.4: An example of a $\mathcal{T}_3$.

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 section 6.7 of [3] for the proof.

2.1.2 Amenable holes.

Theorem 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 [3] for the proof.

With theorem 2.3 we can describe the rest of the algorithm.

Algorithm 2.1 (List possible near cleaners, 9.2 of [3])
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 2.1 we will need the following theorem.

Theorem 2.4 (9.1 of [3])**.** *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 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 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 2.1 is correct.

Algorithm 2.2 (Test possible near clener, 5.1 of [3])
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)
- $r(x_3, y_1), r(x_3, y_2) \geq n$

End

Finding and Using near-cleaners.

Overview of proof of why algorithm using Half-Cleaners is correct.

2.2 Implementation

Anything interesting about algo/data structure?

2.2.1 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.

2.3 Parallelism with CUDA (?)

TODO

2.4 Experiments

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

Description of tests used.

Results and Corollary - almost usable algorithm.

Chapter 3

Coloring Berge Graphs

3.1 Ellipsoid method

Description.

Implementation.

Experiments and results.

3.2 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] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.