Jagiellonian University
Department of Theoretical Computer Science

Adrian Siwiec

Perfect Graph Recognition and Coloring

Master Thesis

Supervisor: dr inż. Krzysztof Turowski

September 2020

Abstract

TODO

Contents

1 Perfect Graphs

All graphs in this paper are finite, undirected and have no loops or parallel edges. We denote the chromatic number of graph G by $\chi(G)$ and the cardinality of the largest clique of G by $\omega(G)$. *Coloring* of a graph means assigning every node of a graph a color. A coloring is *valid* iff every two nodes sharing an edge have different colors. An *optimal* coloring (if exists) is a valid coloring using only $\omega(G)$ colors.

Given a graph $G = (V, E)$, sometimes by $V(G)$ and $E(G)$ we will denote a set of nodes and edges of G. Given a set $X \subseteq V$ by $G[X]$ we will denote a graph induced on X. A graph G is *perfect* iff for all $X \subseteq V(G)$ we have $\chi(G[X]) = \omega(G[X])$.

Definitions and conventions

Where should we put it? Should we put all definitions here, or leave them in text?

- By solid lines we will mark edges, by dashed lines we will mark nonedges, when significant. Sometimes nonedges will not be marked in order not to clutter the image.
- A node is X-complete in G iff it is adjacent to all nodes in X.
- Given a path P, by P^* we will denote its inside.

Give some examples why perfect graphs are interesting, some subclasses, and problems that are solvable for perfect graphs, including recognition and coloring

Given a graph G, its *complement* $\overline{G}$ is a graph with the same vertex set and in which two distinct nodes u, v are connected in $\overline{G}$ iff they are not connected in G. For example a clique in a graph becomes an independent set in its complement. A perfect graph theorem, first conjured by Berge in 1961 [1] and then proven by Lovász in 1972 [5] states that a graph is perfect iff its complement graph is also perfect.

Should we give some proof of that here? Maybe based on proof in [4]

A *hole* is an induced chordless cycle of length at least 4. An *antihole* is an induced subgraph whose complement is a hole. A *Berge* graph is a graph with no holes or antiholes of odd length.

In 1961 Berge conjured that a graph is perfect iff it is Berge 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" [3]. The following overview of the proof will be based on this paper and on an article withe the same name by Cornuéjols [4].

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 $\frac{n-1}{2}$. It is therefore clear, that if a graph is not Berge it is not perfect. To prove that every Berge graph is perfect is the proper part of the strong perfect graph theorem.

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

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 *path* in G is an induced subgraph that is connected, with at least one node, no cycle and no node of degree larger than 2 (sometimes called chordless path). The *length* of a path or a cycle is the number of edges in it. A *triangle* in a graph is a set of three pairwise adjacent nodes.

Should we move these definitions elsewhere?

A *pyramid* in G is an induced subgraph formed by the union of a triangle $\{b_1, b_2, b_3\}$, three paths $\{P_1, P_2, P_3\}$ and another node 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 node 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 $\{b_1, b_2, b_3\}$.

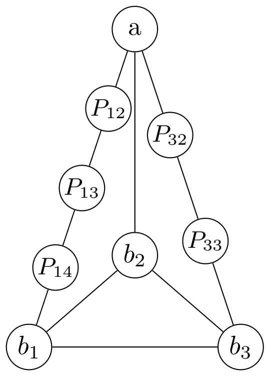


Figure 1: An example of a pyramid.

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.

We will give an algorithm finding pyramid, but first we need some additional definitions. If K is a pyramid $(a, b_1, b_2, b_3, P_1, P_2, P_3)$ we say its *frame* is the 10-tuple iff

finish it up

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 node 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 $O(|V(G)|^6)$ sextuples, and it takes $O(|V(G)|)$ time to check each one. For each such sextuple 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-node path with node 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 sextuple 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 nodes of M that either belong to $P_1(m_1)$ or have a neighbor in $P_1(m_1)$. Color all other nodes 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 nodes. 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 nodes $v_1, \ldots, v_5$ and a path P is a *jewel* iff:

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

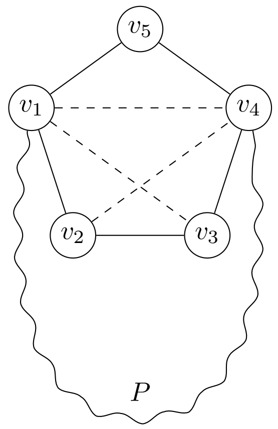


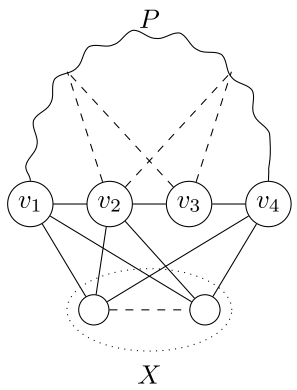
Figure 2: An example of a jewel.

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 node to complete the hole. See paragraph 2.2.1 for more implementation details.

Configurations of type $\mathcal{T}_2$ A configuration of type $\mathcal{T}_2$ is a six $(v_1, v_2, v_3, v_4, P, X)$, 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 nodes.
- 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.

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 nodes 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 Roussel-Rubio lemma [6] as stated in [2] that says:

Lemma 2.1 (Roussel-Rubio Lemma)**.** *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 nodes of P with interior in X.*

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 nodes 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 [3], 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 $v_1, \dots, v_6$, P, X, such that:

- $v_1, \dots v_6$ are distinct nodes.
- 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 nodes, 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 node 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.

picture?

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 nodes 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 nodes. 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 node of $F' \setminus \{v_5\}$ is X-complete. Let F be the union of F' and the set of all X-complete nodes 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 node 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 node 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 [2] for the proof.

Should we include it?

2.1.2 Amenable holes.

For a shortest odd hole C in G, we will call a node $v \in V(G) \setminus V(C)$ *C-major* iff the set of its neighbors in C is not contained in any 3-node path of C. An odd hole C will be called *clean* if no vertex is C-major.

A hole C of G is *amenable* iff C is a shortest odd hole in G of length at least 7, and for every anticonnected set X of C-major nodes, there is an X-complete edge in C (an edge is X-complete iff both its ends are X-complete).

The rest of the algorithm rests upon the following theorem:

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.*

Any ideas on what else to say here? List all 9 steps?

For a shortest odd hole C in G, a subset X of $V(G)$ is a *near-cleaner for C* iff X contains all C-major nodes and $X \cap V(C)$ is a subset of node set of some 3-node path of C.

Finding and Using Half-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 node to complete the hole - much faster than iterating nodes.

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.

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

A Perfect Graph Coloring algorithm

TODO

References

[1] 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.

[2] 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`.

[3] 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`.

[4] 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`.

[5] 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`.

[6] 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`.