ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΜΑΘΗΜΑΤΙΚΩΝ
ΤΟΜΕΑΣ ΕΦΑΡΜΟΣΜΕΝΩΝ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΜΑΘΗΜΑΤΙΚΩΝ

Αθανάσιος Ζήσης

Αλγοριθμοι για τον υπολογισμο κομβων αποφυγης και μονοπατιων αποφυγης

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΑΤΡΙΒΗ

Ιωάννινα, 2021

UNIVERSITY OF IOANNINA
FACULTY OF SCIENCE
DEPARTMENT OF MATHEMATICS
SECTION OF APPLIED AND
COMPUTATIONAL MATHEMATICS

Athanasios Zisis

Algorithms for Computing Avoidable Vertices and Avoidable Paths

---

MASTER THESIS

Ioannina, 2021

Dedicated to the memory of my loving sister Irene
and to the memory of my parents Evangelos and Eleni.

''Μικρού δ' αγώνος ου μέγα έρχεται κλέος.''
Σοφοκλής, 496-406 π.Χ.
(Από ασήμαντο αγώνα δεν προκύπτει μεγάλη δόξα.)

"Little effort will not bring much glory."
Sophocles, 496-406 B.C.

Η παρούσα Μεταπτυχιακή Διατριβή εκπονήθηκε στο πλαίσιο των σπουδών για την απόκτηση του Μεταπτυχιακού Διπλώματος Ειδίκευσης στα ''Εφαρμοσμένα Μαθηματικά και Πληροφορική'' που απονέμει το Τμήμα Μαθηματικών του Πανεπιστημίου Ιωαννίνων.

Εγκρίθηκε την 02/12/2021 από την εξεταστική επιτροπή:

Ονοματεπώνυμο	**Βαθμίδα**
Χάρης Παπαδόπουλος	Αναπληρωτής Καθηγητής (Επιβλέπων)
Λουκάς Γεωργιάδης	Αναπληρωτής Καθηγητής
Λεωνίδας Παληός	Καθηγητής

ΥΠΕΥΘΥΝΗ ΔΗΛΩΣΗ

"Δηλώνω υπεύθυνα ότι η παρούσα διατριβή εκπονήθηκε κάτω από τους διεθνείς ηθικούς και ακαδημαϊκούς κανόνες δεοντολογίας και προστασίας της πνευματικής ιδιοκτησίας. Σύμφωνα με τους κανόνες αυτούς, δεν έχω προβεί σε ιδιοποίηση ξένου επιστημονικού έργου και έχω πλήρως αναφέρει τις πηγές που χρησιμοποίησα στην εργασία αυτή."

Αθανάσιος Ζήσης

Ευχαριστιες

Με την ολοκλήρωσή της παρούσας Μεταπτυχιακής Διατριβής θα ήθελα να απευθύνω ένα βαθύ ευχαριστώ σε αυτούς που με βοήθησαν να την φέρω σε πέρας.

Θα ήθελα να ευχαριστήσω ιδιαίτερα τον επιβλέποντά μου, Αναπληρωτή Καθηγητή κύριο Χάρη Παπαδόπουλο, ο οποίος προσέφερε το ενδιαφέρον θέμα της εργασίας αυτής και μου παρείχε σε όλα τα στάδια της εξέλιξης της Μεταπτυχιακής μου Διατριβής, παρόλο τον περιορισμένο χρόνο του, όλη την απαραίτητη καθοδήγηση, αλλά και οτιδήποτε άλλο χρειαζόμουν.
Επίσης, τον ευχαριστώ θερμά, που συνετέλεσε με κάθε δυνατό τρόπο να εδραιώσουμε μεταξύ μας μια άψογη, υψηλής ποιότητας και επαγγελματική ερευνητική συνεργασία, βασισμένη στον αλληλοσεβασμό.

Επιπλέον, θα ήθελα να ευχαριστήσω τα μέλη της εξεταστικής επιτροπής, τους κυρίους Λουκά Γεωργιάδη και Λεωνίδα Παληό, για τα χρήσιμα σχόλια και τις παρατηρήσεις τους.

Αισθάνομαι επίσης την ανάγκη να ευχαριστήσω θερμά την συνάδελφο και συμφοιτήτριά μου στο Μεταπτυχιακό Πρόγραμμα Σπουδών, κυρία Ευαγγελία Τακαντζιά, για την αμέριστη συμπαράστασή της και την πολύτιμη συνδρομή της προς το πρόσωπο μου, σε όλες τις δύσκολες στιγμές, ακαδημαϊκές και μη, της συνολικής μάλιστα διάρκειας των μεταπτυχιακών μου σπουδών.

Τέλος, θα ήταν παράλειψή μου να μην ευχαριστήσω την φίλη, κυρία Ολυμπία Φώτου, για την πολύπλευρη στήριξή της κατά την διάρκεια των μεταπτυχιακών μου σπουδών.

Περιληψη

Η κορυφή ενός γραφήματος της οποίας η γειτονιά είναι κλίκα, καλείται simplicial. Είναι γνωστό ότι η εύρεση όλων των simplicial κορυφών ενός γραφήματος με n κορυφές και m ακμές μπορεί να επιτευχθεί σε $O(nm)$ χρόνο ή $O(n^\omega)$ χρόνο, όπου $O(n^\omega)$ είναι ο χρόνος που απαιτείται για τον πολλαπλασιασμό πινάκων διάστασης $n \times n$. Η έννοια των κορυφών αποφυγής (avoidable vertices) αποτελεί γενίκευση της έννοιας των simplicial κορυφών με τον ακόλουθο τρόπο: μια κορυφή u είναι κορυφή αποφυγής (avoidable) εαν κάθε επαγόμενο μονοπάτι τριών κορυφών με μεσαία κορυφή την u περιέχεται σε έναν επαγόμενο κύκλο.

Παρουσιάζουμε και αναλύουμε αλγόριθμους που σχεδιάσαμε για την εύρεση όλων των κορυφών αποφυγής ενός γραφήματος μέσω των εννοιών των minimal triangulations και της ανίχνευσης κοινής γειτονιάς. Πιο συγκεκριμένα, δίνουμε αλγόριθμους με πολυπλοκότητα χρόνου $O(n^2m)$ και $O(n^{1+\omega})$, αντίστοιχα. Επιπλέον, προτείνουμε και έναν πιο γρήγορο αλγόριθμο με χρονική πολυπλοκότητα $O(n^2 + m^2)$, η οποία συμπίπτει με την αντίστοιχη πολυπλοκότητα χρόνου της εύρεσης των simplicial κορυφών σε αραιά γραφήματα όπου χαρακτηρίζονται από $m = O(n)$.

Αξίζει να σημειώσουμε ότι κατά την ανάλυση των συγκεκριμένων αλγορίθμων σχεδιάσαμε επιπλέον έναν βέλτιστο γραμμικό αλγόριθμο για τα cograph και έναν αλγόριθμο για τα chordal γραφήματα με πολυπλοκότητα χρόνου $O(nm)$.

Επεκτείνουμε τα αποτελέσματά μας, εξετάζοντας τις ακμές αποφυγής (avoidable edges) και, πιο γενικά, τα μονοπάτια αποφυγής (avoidable paths) καθώς αποτελούν την φυσική γενίκευση των κορυφών αποφυγής. Παρουσιάζουμε έναν αλγόριθμο χρονικής πολυπλοκότητας $O(nm)$, ο οποίος αναγνωρίζει πότε ένα δωθέν επαγόμενο μονοπάτι (ή ακμή) είναι μονοπάτι (ή ακμή) αποφυγής.

Τέλος, συγκρίνουμε την εμπειρική απόδοση των προτεινόμενων αλγορίθμων μας, που εκτελούνται στο συνολικό γράφημα εισόδου, καθώς και στα συρικνωμένα υπογραφήματά του. Για το σκοπό αυτό, διεξάγουμε μια διεξοδική πειραματική μελέτη για να αναδείξουμε τα πλεονεκτήματα και τις αδυναμίες κάθε προτεινόμενης τεχνικής.

Abstract

A simplicial vertex of a graph is a vertex whose neighborhood is a clique. It is known that listing all simplicial vertices can be done in $O(nm)$ time or $O(n^\omega)$ time, where $O(n^\omega)$ is the time needed to perform a fast matrix multiplication. The notion of avoidable vertices generalizes the concept of simplicial vertices in the following way: a vertex u is avoidable if every induced path on three vertices with middle vertex u is contained in an induced cycle.

We present algorithms for listing all avoidable vertices of a graph through the notion of minimal triangulations and common neighborhood detection. In particular we give algorithms with running times $O(n^2m)$ and $O(n^{1+\omega})$, respectively. Additionally, we propose a faster algorithm that runs in time $O(n^2 + m^2)$, and thus matches the corresponding running time of listing the simplicial vertices on sparse graphs with $m = O(n)$.

Moreover, our results imply an optimal algorithm on cographs and $O(nm)$-time algorithm on chordal graphs.

To complement our results, we consider their natural generalizations of avoidable edges and avoidable paths. We propose an $O(nm)$-time algorithm that recognizes whether a given induced path is avoidable.

Finally, we compare the empirical performance of our proposed algorithms that are performed on the overall input graph as well as on its contracted subgraphs. To that end, we conduct a thorough experimental study to highlight the merits and weaknesses of each technique.

Περιεχομενα

CHAPTER 1

Introduction

Closely related to chordal graphs is the notion of a simplicial vertex, that is a vertex whose neighborhood induces a clique. In particular, Dirac [13] proved that every chordal graph admits a simplicial vertex. However not all graphs contain a simplicial vertex. Due to their importance to several algorithmic problems, such as finding a maximum clique or detecting the chromatic number, it is natural to seek for fast algorithms that list all simplicial vertices of a graph. For doing so, the naive approach takes $O(nm)$ time, whereas the fastest algorithms take advantage of computing the square of an $n \times n$ binary matrix and run in $O(n^\omega)$ and $O(m^{2\omega/(\omega+1)})$ time [15]. Hereafter we assume that we are given a graph G on n vertices and m edges; currently, $\omega < 2.37286$ [2].

A natural way to generalize the concept of simplicial vertices is the notion of an avoidable vertex. A vertex u is avoidable if either there is no induced path on three vertices with middle vertex u, or every induced path on three vertices with middle vertex u is contained in an induced cycle. Thus every simplicial vertex is avoidable, however the converse is not necessarily true. As opposed to simplicial vertices, it is known that every graph contains an avoidable vertex [1, 7, 5, 17]. Extending the notion of avoidable vertices is achieved through avoidable edges and, more general, avoidable paths. This is accomplished by replacing the middle vertex in an induced path on three vertices by an induced path on arbitrary $k \geq 2$ vertices, denoted by P_k. Beisegel et. al [3] proved first that every non-edgeless graph contains an avoidable edge, considering the case of $k = 2$. Regarding the existence of an avoidable induced path of arbitrary length, Bonamy et al. [9] settled a conjecture in [3] and showed that every graph is either P_k-free or contains an avoidable P_k.

Since avoidable vertices generalize simplicial vertices, it is expected that avoidable vertices find applications in further algorithmic problems. Indeed, Beisegel et. al [3] revealed new polynomially solvable cases of the maximum

weight clique problem that take advantage the notion of avoidable vertices. Similar to simplicial vertices, the complexity of a problem can be reduced by removing avoidable vertices, tackling the problem on the reduced graph. It is therefore of interest to list all avoidable vertices efficiently. If we are only interested in computing two avoidable vertices this can be done in linear time by using fast graph searches [5, 3]. However, computing the set of all avoidable vertices requires to decide for each vertex of the graph whether it is avoidable and a usual graph search cannot guarantee to test all vertices.

A naive approach that recognizes of a single vertex u of a graph G whether it is avoidable or not, needs to check if all neighbors of u are pairwise connected in an induced subgraph of G. Thus the running time of recognizing an avoidable vertex is $O(n^3 + n^2m)$ or, as explicitly stated in [3], it can be expressed as $O(\overline{m}\cdot(n+m))$ where $\overline{m}$ is the number of edges in the complement of G. Inspired by both running times, we first show that we can reduce in linear time the listing problem on a graph G having $m \geq n$ and $\overline{m} \geq n$. In a sense such a result states that graphs that are sparse ($m < n$) or dense ($\overline{m} < n$) can be decomposed efficiently to smaller connected graphs for which their complement is also connected. Towards this direction, we give an interesting connection with the avoidable vertices on the complement of G. As a result, the naive algorithms for listing all avoidable vertices take $O(n^3\cdot m)$ and $O(n\cdot\overline{m}\cdot m)$ time, respectively. Moreover, based on the proposed reduction we derive an optimal, linear-time, algorithm for listing all avoidable vertices on graphs having no induced path on four vertices, known as cographs.

Our main results consist of new algorithms for listing all avoidable vertices in running times comparable to the ones for listing simplicial vertices. More precisely, we propose three main approaches that result in algorithms for listing all avoidable vertices of a graph G with the following running times:

- $O(n^2 \cdot m)$, by using a minimal triangulation of G. A close relationship between avoidable vertices and minimal triangulation was already known [3]. However, listing all avoidable vertices through the proposed characterization is inefficient, since one has to produce *all* possible minimal triangulations of G. Here we strengthen such a characterization in the sense that it provides an efficient recognition based on one particular minimal triangulation of G. More precisely, we take advantage of vertex-incremental minimal triangulations that can be computed in $O(nm)$ time [8].

- $O(n^2 + m^2)$, by exploring structural properties on each edge of G. This

approach is based on a modified, traditional breadth-first search algorithm. Our task is to construct search trees rooted at a particular vertex that reach all vertices of a predescribed set S, so that every non-leaf vertex does not belong to S. If such a tree exists then every path from the root to a leaf that belongs to S is called an S-excluded path. It turns out that S-excluded paths can be tested in linear time and we need to make $2m$ calls of a modified breadth-first search algorithm.

- $O(n^{1+\omega})$, where $O(n^{\omega})$ is the running time for matrix multiplication. For applying a matrix multiplication approach, we contract the connected components of G that are outside the closed neighborhood of a vertex. Then we observe that a vertex u is avoidable if the neighbors of u are pairwise in distance at most two in the contracted graph. As the distance testing can be encapsulated by the square of its adjacency matrix, we deduce an algorithm that takes advantage of a fast matrix multiplication.

We should note that each of the stated algorithms is able to recognize if a given vertex u of G is avoidable in time $O(nm)$, $O(d(u)(n+m))$, and $O(n^{\omega})$, respectively, where $d(u)$ is the degree of u in G. Further, all of our proposed algorithms are characterized by their simplicitiy and, besides the fast matrix multiplication, consist of basic ingredients that avoid using sophisticated data structures.

In addition, we consider the natural generalizations of avoidable vertices, captured within the notions of the avoidable edges and avoidable paths. A naive algorithm that recognizes an avoidable edge takes time $O(n^2 \cdot m)$ or $O(\overline{m} \cdot m)$. Here we show that recognizing an avoidable edge of a graph G can be done in $O(n \cdot m)$ time. This is achieved by taking advantage of the notions of the S-excluded paths and their efficient detection by the modified breadth-first search algorithm. Also notice that an avoidable edge is an avoidable path on two vertices. We are able to reduce the problem of recognizing an avoidable path of arbitrary length to the recognition of an avoidable edge. In particular, given an induced path we prove that we can replace the induced path by an edge and test whether the new added edge is avoidable or not in a reduced graph. Therefore our recognition algorithm for testing whether a given induced path is avoidable takes $O(n \cdot m)$ time.

1.1 Road Map

In Chapter 2 we give the necessary general notation and the basic definitions of avoidable vertices.

In Chapter 3 we consider separately sparse and dense graphs. Moreover we give an optimal algorithm on cographs with running time $O(n + m)$.

In Chapter 4 we compute avoidable vertices using minimal triangulations and we propose a fast algorithm that uses a new notion, namely the $S-$*excluded path* and *protecting*. Moreover we give a $O(nm)$-time algorithm on chordal graphs.

In Chapter 5 we compute avoidable vertices via contractions and fast matrix multiplication.

In Chapter 6 we propose a fast algorithm for recognizing avoidable edges and paths using the notion of *protected edge*.

In Chapter 7 we conduct a thorough experimental analysis of the proposed algorithms for listing all avoidable vertices of a graph.

CHAPTER 2

Preliminaries

All graphs considered here are finite undirected graphs without loops and multiple edges. We refer to the textbook by Bondy and Murty [10] for any undefined graph terminology. For a graph $G = (V_G, E_G)$, we use V_G and E_G to denote the set of vertices and edges, respectively. We use n to denote the number of vertices of a graph and use m for the number of edges. Given $x \in V_G$, we denote by $N_G(x)$ the neighborhood of x. The *degree* of x is the number of edges incident to x, denoted by $d_G(x)$. That is, $d_G(x) = |N_G(x)|$. The closed neighborhood of x, denoted by $N_G[x]$, is defined as $N_G(x) \cup \{x\}$. For a set $X \subset V(G)$, $N_G(X)$ denotes the set of vertices in $V(G) \setminus X$ that have at least one neighbor in X. Analogously, $N_G[X] = N_G(X) \cup X$. Given $X \subseteq V_G$, we denote by $G - X$ the graph obtained from G by the removal of the vertices of X. If $X = \{u\}$, we also write $G - u$. The *subgraph induced by* X is denoted by $G[X]$, and has X as its vertex set and $\{uv \mid u, v \in X \text{ and } uv \in E_G\}$ as its edge set. For $R \subseteq E(G)$, $G \setminus R$ denotes the graph $(V(G), E(G) \setminus R)$, that is a subgraph of G. If $R = \{e\}$, we also write $G \setminus e$.

A *clique* of G is a set of pairwise adjacent vertices of G, and a *maximal clique* of G is a clique of G that is not properly contained in any clique of G. An *independent set* of G is a set of pairwise non-adjacent vertices of G. The induced path on $k \geq 2$ vertices is denoted by P_k and the induced cycle on $k \geq 3$ vertices is denoted by C_k. For an induced path P_k, the vertices of degree one are called *endpoints*. A vertex v is *universal* in G if $N[v] = V(G)$ and v is *isolated* if $N(v) = \emptyset$. A vertex of degree one is called *leaf*. A graph is *connected* if there is a path between any pair of vertices. A *connected component* of G is a maximal connected subgraph of G. Given two vertices u and v of a connected graph G, a set $S \subset V_G$ is called (u, v)-*separator* if u and v belong to different connected components of $G - S$. We say that S is a separator if there exist two vertices u and v such that S is a (u, v)-separator. For a set of finite graphs $\mathcal{H}$, we say that a graph G is $\mathcal{H}$-free if G does not contain an induced subgraph

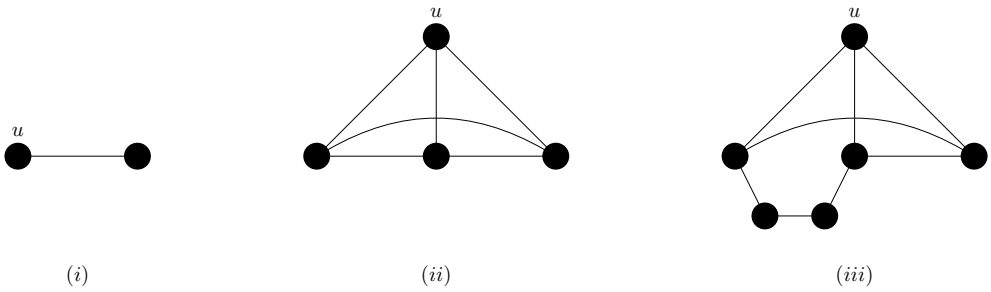

Figure 2.1: (i) Vertex u is simplicial and thus avoidable. (ii) Vertex u is simplicial and thus avoidable. (iii) A non-simplicial avoidable vertex u.

isomorphic to any of the graphs of $\mathcal{H}$.

The *disjoint union* of two graphs G and H, denoted by $G \cup H$, is the graph on vertex set $V(G) \cup V(H)$ and edge set $E(G) \cup E(H)$. The *complement* of G, denoted by $\overline{G}$, is the graph on vertex set $V(G)$ and edge set $\{uv \mid uv \notin E(G)\}$. We say that a graph G is *co-connected* if $\overline{G}$ is connected. Moreover a *co-component* of G is a connected component of $\overline{G}$.

Given an edge $e = xy$, the *contraction* of e removes both x and y and replaces them by a new vertex w, which is made adjacent to those vertices that were adjacent to at least one of the vertices x and y, that is $N(w) = (N(x) \cup N(y)) \setminus \{x, y\}$. Let S be a vertex set of G such that $G[S]$ is connected. If we repeatedly contract an edge of $G[S]$ until one vertex remains in S then we say that we *contract S into a single vertex*. In different terminology, *contracting a set of vertices S* is the operation of substituting the vertices of S by a new vertex w with $N(w) = N(S)$.

A vertex v is called *simplicial* if the vertices of $N_G(v)$ induce a clique (see Figure 2.1 $(i) - (ii)$). Listing all simplicial vertices of a graph can be done $O(nm)$ time. The fastest algorithm for listing all simplicial vertices takes time $O(n^\omega)$, where $O(n^\omega)$ is the time needed to multiply two $n \times n$ binary matrices [15] (currently, $\omega < 2.37286$ [2]). Avoidable vertices and edges generalize the concept of simplicial vertices in a natural way.

Definition 1. A vertex v is called *avoidable* if every P_3 with middle vertex v is contained in an induced cycle. Equivalently, v is avoidable if $d_G(v) \leq 1$ or for every pair $x, y \in N_G(v)$ the vertices x and y belong to the same connected component of $G - (N_G[u] \setminus \{x, y\})$ (see Figure 2.1 (iii)).

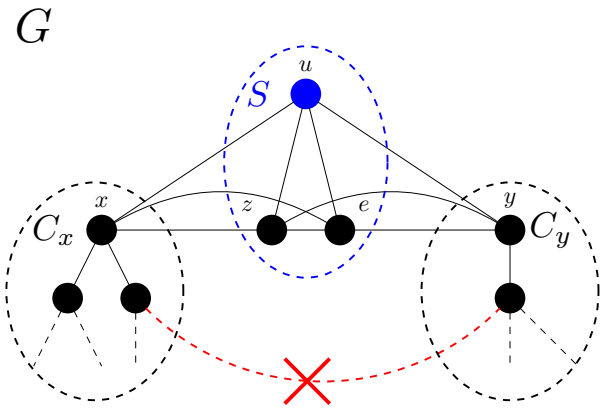

Figure 2.2: A connected graph G. A non-avoidable vertex u in G has a $(x, y) - separator$ S such that $S \subset N_G[u]$ for some vertices $x, y \in N_G(u)$.

Every simplicial vertex is avoidable, however the converse is not necessarily true. It is known that every graph contains an avoidable vertex [1, 7, 17]. Every vertex of a graph of degree ≤ 1 is simplicial and hence avoidable. Thus a non-avoidable vertex of a graph, has degree ≥ 2.

Observation 1. *Let G be a graph and let u be a vertex of G. Then u is non-avoidable if and only if there is an (x, y)-separator S that contains u such that $S \subset N_G[u]$ for some vertices $x, y \in N_G(u)$.*

Proof. Assume that u is non-avoidable. Then by Definition 1, there are two vertices $x, y \in N_G(u)$ that belong to different connected components in $G - (N_G[u] \setminus \{x, y\})$. This means that $S = N_G[u] \setminus \{x, y\}$ is an (x, y)-separator (see Figure 2.2). On the other hand, if there is such a separator S for some vertices $x, y \in N_G(u)$ then x and y do not belong to the same connected component in the graph $G - S$ and, consequently, also in the graph $G - (N_G[u] \setminus \{x, y\})$, because $S \subset N_G[u]$. Thus u is non-avoidable vertex. □

CHAPTER 3

Detecting Avoidable Vertices in Sparse or Dense Graphs

Here we show how to compute efficiently all avoidable vertices on sparse or dense graphs. In particular, for a graph G on n vertices and m edges, we consider the cases in which $m < n$ (sparse graphs) or $\overline{m} < n$ (dense graphs), where $\overline{m} = |E(\overline{G})|$. Our main motivation comes the naive algorithm that lists all avoidable vertices in $O(n \cdot \overline{m} \cdot (n + m))$ time that takes advantage of the non-edges of G [3]. We will show that we can handle such cases in linear time, so that the running time of the naive algorithm can be written as $O(n^3 \cdot m)$. For doing so, we consider the impact of avoidable vertices on the complement of a graph by considering the connected components in both G and $\overline{G}$. Before reaching the details of our approach, we give a simple linear-time algorithm on the class of cographs, since they can be totally decomposed by the corresponding operations.

3.1 Appetizer: an optimal algorithm on cographs

A graph G is *cograph* if every induced subgraph of G on at least two vertices is either disconnected or its complement is disconnected. Cographs are exactly the class of P_4-free graphs [11]. Every cograph G admits a unique tree representation known as *cotree* which is a rooted tree T with two types of internal nodes: 0-nodes and 1-nodes. The vertices of G are assigned to the leaves of T in a one-to-one manner. Thus T contains $O(n)$ nodes (see Figure 3.1). The properties of a cotree T are summarized as follows:

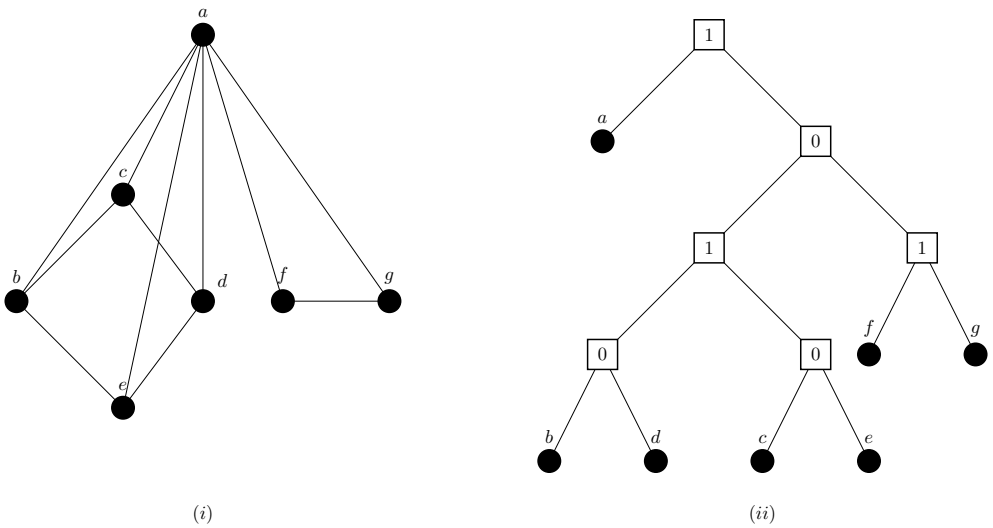

Figure 3.1: Illustrating a cograph and the corresponding cotree.

(i) Two vertices of G are adjacent if and only if their least common ancestor in T is a 1-node.

(ii) Every internal node of T has at least two children.

(iii) No two internal nodes of the same type are adjacent in T.

The cotree of a cograph is unique and can be generated in linear time [12].

We give the following characterization of avoidable vertices in G in terms of the cotree T. For doing so, we denote by $p(u)$ the parent of a vertex u in T. A 1-node w of T is called *full 1-node* if the children of w are all leaves in T.

Lemma 2. *Let T be a cotree of a cograph G and let u be a vertex of G. Then, u is avoidable in G if and only if either $p(u)$ is a 0-node or $p(u)$ is a full 1-node.*

Proof. We first introduce some notation. For a node w of T, we let T_w be the subtree of T rooted at w and we denote by $V(T_w)$ the set of leaves in T_w. Recall that $V(T_w)$ corresponds to a subset of vertices of G. By property (i) observe that all the vertices of $V(T_w)$ are either adjacent or non-adjacent to a vertex x of $V(G) \setminus V(T_w)$. Let r be the root of T and let w be the parent

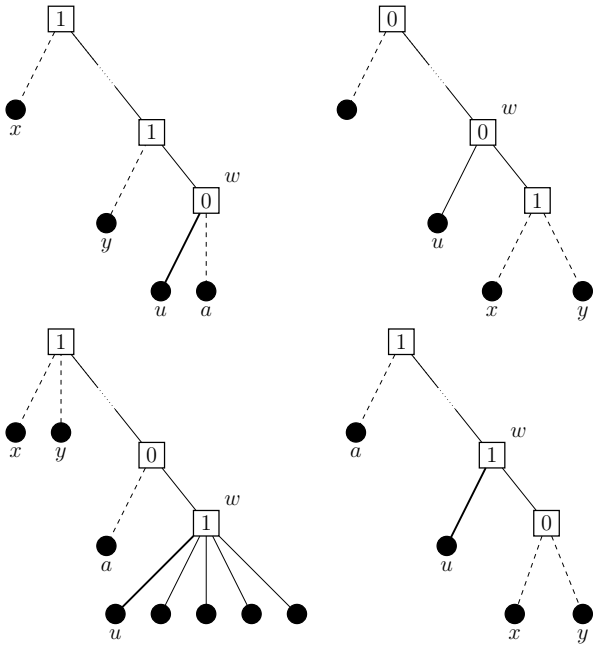

Figure 3.2: Illustrating the cases considered in the proof of Lemma 2.

node of vertex u, that is $w = p(u)$. We consider separately the following cases (see Figure 3.2).

- Assume that w is a 0-node in T. We show that u is avoidable in G. Consider two vertices $x, y \in N_G(u)$. By property (i), $x, y \in V(T_r) \backslash V(T_w)$ and any vertex of $V(T_w)$ is non-adjacent to u. Moreover, property (ii) implies that there is a vertex $a \in V(T_w) \setminus \{u\}$ such that $au \notin E(G)$. Thus, both x and y are adjacent to u and a, since $x, y \notin V(T_w)$. Hence, regardless of whether x and y being adjacent, there is a path between x and y that does not contain any vertex of $N_G(u)$.

- Assume that w is a full 1-node in T. We show that u is avoidable in G. Consider two vertices $x, y \in N_G(u)$. If $x \in V(T_w)$ then $xy \in E(G)$ because either $y \in V(T_w)$ as a leaf vertex, or $y \notin V(T_w)$ and y is adjacent to every vertex of $V(T_w)$ as $uy \in E(G)$. Suppose that both $x, y \in V(T_r) \setminus V(T_w)$. Let $P(r, w)$ be the unique path of T between the root r and the 1-node w. Since $x, y \in N_G(u)$, there are 1-nodes w_x and w_y (not necessarily distinct) on $P(r, w)$ such that $x \in V(T_{w_x})$ and $y \in V(T_{w_y})$. Now consider the parent w' of w in T. By property (iii), w'

exists and is a 0-node of T. Thus there is a vertex $a \in V(T_{w'}) \setminus V(T_w)$ that is non-adjacent to u. Since the least common ancestor of x and a is w_x, by property (i) we have $xa \in E(G)$. Similarly, we have $ya \in E(G)$. Hence there is a path between x and y that contains a non-neighbor of u, which shows that u is an avoidable vertex of G.

- Assume that w is a 1-node that is not full in T. We show that u is non-avoidable in G. Let w' be a non-leaf child of w. By property (iii), w' is a 0-node. Moreover, property (ii) implies that there are vertices $x, y \in V(T_{w'})$ for which their least common ancestor is w'. Thus $xy \notin E(G)$ and $ux, uy \in E(G)$, because w is a 1-node. If there is no path between x and y in $G - u$ then u is non-avoidable. Let A be the internal vertices of an induced path between x and y in $G - u$. Since G is P_4-free, every vertex of A is adjacent to both x and y, so that $A = \{a\}$. We show that u is adjacent to a. To see this, observe that a does not belong to $V(T_{w'})$, since w' is the 0-node that is the least common ancestor of x and y. Hence a belongs to $V(T_r) \setminus V(T_{w'})$ and its least common ancestor w_a with x and y is a 1-node. This means that w_a is an ancestor of w' that is a 1-node in T. As w' is a child of w, we deduce that w_a is the least common ancestor of a and u. Thus $ua \in E(G)$, which means that u is non-avoidable, since there is no path between x and y that avoids any neighbor of u.

Therefore, we have a complete characterization of u since all cases have been considered depending on the parent of u in T. □

Thus, we deduce the following optimal algorithm for the vertices of a cograph G. Note that, given a cograph G, its corresponding cotree T can be constructed in $O(n + m)$ time [12].

Theorem 3. *Given a cotree T of a cograph G, there is an $O(n)$-time algorithm that lists all avoidable vertices of G.*

Proof. We first mark the internal nodes of the cotree T that have as children only leaves of T. By a single bottom-up traversal from the leaves of T, this can be done in $O(n)$ time. Thus applying Lemma 2 in a straightforward way on the cotree T with the marked information, results in an $O(n)$-time algorithm. □

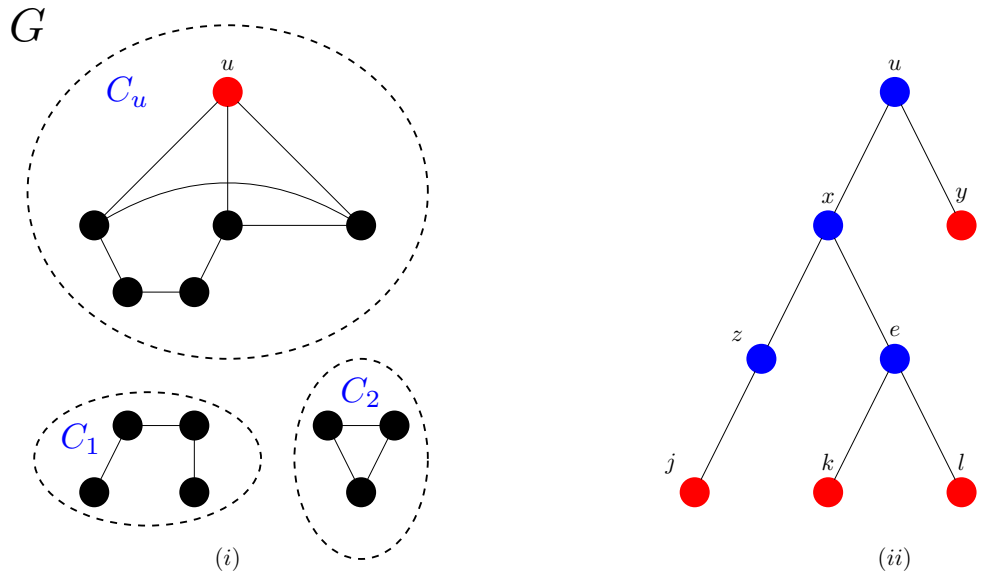

Figure 3.3: (i) Avoidability of a vertex u in a graph G is relevant only with the component $C_u \subseteq G$ of u. (ii) Trees have a trivial solution because only their leafs are avoidable vertices.

3.2 Sparse or dense graphs

Here we extend the previous notions on cographs and show how to handle the cases in which $m < n$ (sparse graphs) or $\overline{m} < n$ (dense graphs).

It is not difficult to handle sparse graphs. Observe that $m < n$ implies that G is disconnected or G is a tree. The connectedness assumption of the input graph G follows from the fact that a vertex u is avoidable in G if and only if u is avoidable in the connected component containing u, since there are no paths between vertices of different components. Moreover, trees have a trivial solution as the leaves are exactly the set of avoidable vertices (see Figure 3.3). We include both properties in the following statement.

Observation 4. *Let u be a vertex of G and let $C(u)$ be the connected component of G containing u. Then u is avoidable if and only if u is avoidable in $G[C(u)]$. Moreover, if G is a tree then u is avoidable if and only if u is a leaf in G.*

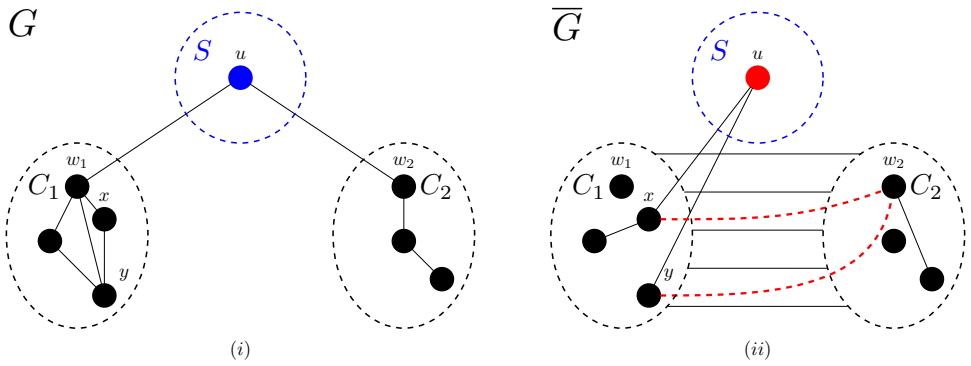

Figure 3.4: Illustrating a non-avoidable vertex u of a graph G, and the components C_1,C_2 and the separator S according to Observation 1. The horizontal lines between C_1 and C_2 in $\overline{G}$, indicate that every vertex of C_1 is adjacent to every vertex of C_2. In $\overline{G}$, u is avoidable.

Next we describe that we can follow almost the same approach on the complement of G. For doing so, we first prove the following result which interestingly relates avoidability on G and $\overline{G}$. Note, however, that the converse is not necessarily true.

Lemma 5. *Let G be a graph and let u be a non-avoidable vertex. Then, u is avoidable in $\overline{G}$.*

Proof. Since u is a non-avoidable vertex in G, there is a separator S that contains u such that $S \subset N_G[u]$ by Observation 1. Let $C_1, \ldots, C_k$ be the connected components of $G - S$, with $k \geq 2$. Notice that at least two components of $C_1, \ldots, C_k$ contain a neighbor of u. Without loss of generality, assume that $C_1 \cap N_G(u) \neq \emptyset$ and $C_2 \cap N_G(u) \neq \emptyset$. Consider the complement $\overline{G}$ and let x, y be two neighbors of u in $\overline{G}$. Observe that both x and y do not belong to S, since $S \subset N_G[u]$. Thus $x \in C_i$ and $y \in C_j$, for $1 \leq i, j \leq k$. We show that either $xy \in E(\overline{G})$ or there is a path in $\overline{G}$ between x and y that avoids vertices of $N_{\overline{G}}(u)$. If $i \neq j$ then $xy \in E(\overline{G})$, because every vertex of C_i is adjacent to every vertex of C_j in $\overline{G}$. Suppose that $x, y \in C_i$. If $C_i \neq C_1$ then there is a vertex $w_1 \in C_1 \cap N_G(u)$ such that $w_1u \notin E(\overline{G})$ and $w_1x, w_1y \in E(\overline{G})$. If $C_i = C_1$ then there is a vertex $w_2 \in C_2 \cap N_G(u)$ such that $w_2u \notin E(\overline{G})$ and $w_2x, w_2y \in E(\overline{G})$ (see Figure 3.4). Thus in both cases there is a path of length two between x and y that avoids vertices $N_{\overline{G}}(u)$. Therefore, u is avoidable in $\overline{G}$. □

We next deal with the case in which $\overline{G}$ is disconnected. Notice that if $G = K_n$ then every vertex of G is simplicial and thus avoidable.

Lemma 6. *Let $G \neq K_n$, $u \in V(G)$, and let $\overline{C}(u)$ be the co-component containing u. Then, u is avoidable in G if and only if $|\overline{C}(u)| > 1$ and u is avoidable in $G[\overline{C}(u)]$.*

Proof. Assume first that $\overline{C}(u) = \{u\}$. Then u is universal in G. Since $G \neq K_n$, there are vertices x, y such that $xy \notin E(G)$. As any path between x and y contains a neighbor of u, we deduce that u is non-avoidable. In the following we assume that $|\overline{C}(u)| > 1$. This assumption implies that there is a vertex $a \in \overline{C}(u)$ such that $ua \notin E(G)$. Also notice that every vertex of $\overline{C}(u)$ is adjacent to every vertex of $V(G) \setminus \overline{C}(u)$.

- Suppose that u is avoidable in G. Assume for contradiction that u is non-avoidable in $G[\overline{C}(u)]$. Then there are vertices x, y in $\overline{C}(u)$ such that $x, y \in N_G(u)$, $xy \notin E(G)$, and every path (if it exists) between x and y in $G[\overline{C}(u)]$ contains a neighbor of u. Since $G[\overline{C}(u)]$ is an induced subgraph of G and u is avoidable in G, there is path in G between x and y that contains a vertex z of $V(G) \setminus \overline{C}(u)$ such that $zu \notin E(G)$. Then, however, we reach a contradiction to the fact that every vertex of $\overline{C}(u)$ is adjacent to every vertex of $V(G) \setminus \overline{C}(u)$, so that $zu \in E(G)$ for any such vertex z. Thus u is avoidable in $G[\overline{C}(u)]$.

- Suppose that u is avoidable in $G[\overline{C}(u)]$. We show that u is avoidable in G. Consider two vertices $x, y \in N_G(u)$. If both vertices x, y belong to $\overline{C}(u)$ then the avoidability of u in $G[\overline{C}(u)]$ carries along G, since $G[\overline{C}(u)]$ is an induced subgraph of G. If $x \in \overline{C}(u)$ and $y \in V(G) \setminus \overline{C}(u)$ then $xy \in E(G)$. Now assume that both vertices x, y belong to $V(G) \setminus \overline{C}(u)$. Then the path $\langle x, a, y \rangle$ with $a \in \overline{C}(u)$ and $ua \notin E(G)$ is the desired path between x and y. Thus u is avoidable in G.

Therefore both directions show the claimed statement. □

In general, avoidability is not a hereditary property with respect to induced subgraphs, even when restricted to the removal of non-avoidable vertices. However, as we show next, the removal of universal vertices does not affect the rest of the graph.

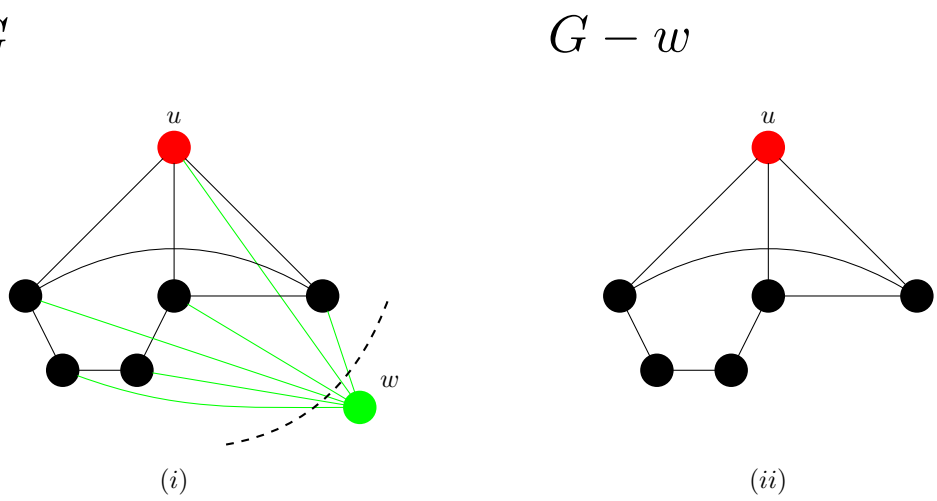

Figure 3.5: Avoidability of a vertex u in a graph G is not affected by the removal of a universal vertex w.

Lemma 7. *Let G be a graph and let w be a universal vertex of G. Then w is avoidable if and only if G is a complete graph. Moreover, any vertex $u \in V(G) \setminus \{w\}$ is avoidable in G if and only if u is avoidable in $G - \{w\}$.*

Proof. First statement follows by Lemma 6 and from the fact that every vertex of a complete graph is simplicial. For the second statement, assume that u is avoidable in G. We show that u is avoidable in the graph $G' = G - \{w\}$. Consider two vertices $x, y \in N_{G'}(u)$. If $xy \in E(G)$ then clearly $xy \in E(G')$. Suppose that $xy \notin E(G)$. Then, as u is avoidable in G, there is a path P between x and y in G. Since w is universal in G, w does not belong to P (see Figure 3.5). Thus P exists in G' which shows that u is avoidable in G'. For the reverse direction, assume that u is avoidable in $G' = G - \{w\}$. Observe that any two vertices $x, y \in N_G(u) \setminus \{w\}$ fulfill the necessary conditions in G, since G' is as induced subgraph of G. Moreover, $w \in N_G(u)$ and for any vertex $x \in N_G(u) \setminus \{w\}$, we have $wx \in E(G)$. Therefore u remains avoidable in G. □

To conclude the cases for which $\overline{m} < n$, we next consider graphs whose complement is a tree. By Observation 4 we restrict ourselves on connected graphs.

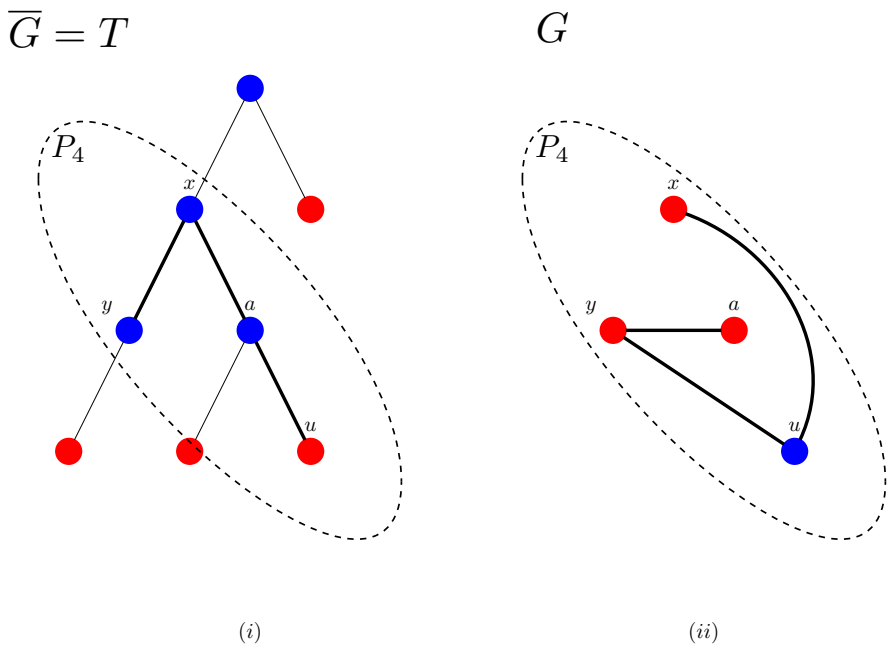

Figure 3.6: Every leaf, and thus avoidable vertex u of $\overline{G} = T$, belongs to a P_4 since both G and $\overline{G}$ are connected and therefore considering this P_4 we deduce that u is non-avoidable in G because of x, y. Also by Lemma 5 we conclude that every non-avoidable and thus non-leaf vertex of $\overline{G}$, is avoidable in G.

Lemma 8. *Let G be a connected graph such that $\overline{G}$ is a tree T. A vertex u of G is avoidable if and only if u is a non-leaf vertex in T.*

Proof. We consider the vertices of T. Let u be a non-leaf vertex of T. Then u is a non-avoidable vertex in $\overline{G}$. Thus by Lemma 5 u is avoidable in G.

Now assume that u is a leaf vertex of T, and thus avoidable in $\overline{G}$. We prove that u is non-avoidable in G. Since both graphs G and $\overline{G}$ are connected, u belongs to a P_4 in T [11]. Let $\langle u, a, x, y\rangle$ be a P_4 in T that contains u. Observe that u is adjacent to every vertex of $V(G) \setminus \{a\}$ in G. Consider the vertices x and y of the P_4 for which $x, y \in N_G(u)$. As $xy \in E(\overline{G})$, we have $xy \notin E(G)$. We show that there is no path between x and y that avoids any neighbor of u in G. If there is a path between x and y then it contains the vertex a and it has the form $\langle x, a, y\rangle$ in G. Then, however, notice that $ya \in E(G)$ but $xa \notin E(G)$ by the induced $P_3 = \langle a, x, y\rangle$ in $\overline{G}$. Thus u is non-avoidable in G,

because of x and y (see Figure 3.6). Therefore, every avoidable vertex of T is non-avoidable in G, since the set of leaves in T are exactly the set of avoidable vertices in $\overline{G}$. □

3.3 A modular decomposition approach

Based on the previous results, we can reduce our problem to a graph G that is both connected and co-connected and neither G nor $\overline{G}$ are isomorphic to trees. To achieve this in linear time we apply known techniques that avoid computing explicitly the complement of G, since we are mainly interested in recursively detecting the components and co-components of G. Such a decomposition, known as the *modular decomposition*, can be represented by a tree structure, denoted by $T(G)$, of $O(n)$ size and can be computed in linear time [16, 19]. More precisely, the leaves of $T(G)$ correspond to the vertices of G and every internal node w of $T(G)$ is labeled with three distinct types according to whether the subgraph of G induced by the leaves of the subtree rooted at w is (i) not connected, or (ii) not co-connected, or (iii) connected and co-connected (see Figure 3.7).

Moreover the connected components and the co-components of types (i) and (ii), respectively, correspond to the children of w in $T(G)$. Let $\mathcal{G}$ be a collection of maximal vertex-disjoint induced subgraphs of G that are both connected and co-connected. Then $T(G)$ determines all graphs of $\mathcal{G}$ in linear time. Observe that if $\mathcal{G}$ is empty, then G is a cograph. In addition, we call $\mathcal{G}$, *typical collection* of G if for each graph $H \in \mathcal{G}$:

- H is connected and co-connected,
- $|V(H)| \leq |E(H)|$, $|V(H)| \leq |E(\overline{H})|$, and
- every avoidable vertex in H is an avoidable vertex in G.

The results of this section deduce the following algorithm.

Theorem 9. *Let G be a graph and let $A(G)$ be the set of avoidable vertices in G. There is a linear-time algorithm, that*

- *computes a typical collection $\mathcal{G}$ of maximal vertex-disjoint induced subgraphs of G and*
- *for every vertex $v \in V(G) \setminus V(\mathcal{G})$, decides if $v \in A(G)$.*

Proof. We first compute $T(G)$ in linear time [16, 19]. Then we visit all nodes of $T(G)$ starting from the root and move towards the leaves of $T(G)$. We stop each branch when we reach either a leaf for which we include it in $A(G)$, or when we reach a graph of $\mathcal{G}$. Given a node w of $T(G)$, let G_w be the graph induced by the leaves of the subtree rooted at w. At each node of $T(G)$ we perform the following steps.

1. If G_w is disconnected then consider the connected components $C_1, \cdots, C_k$ of G by Observation 4. That is, $A(G_w) = A(C_1) \cup \cdots \cup A(C_k)$.
2. If $\overline{G_w}$ is disconnected then consider the co-components $\overline{C}_1, \ldots, \overline{C}_k$ of G such that $|\overline{C}_i| \geq 2$, for each $1 \leq i \leq k$.
   (a) If $G_w = K_n$ (that is, $k = 0$) then $A(G_w) = V(G_w)$.
   (b) Otherwise, $A(G_w) = A(\overline{C}_1) \cup \cdots \cup A(\overline{C}_k)$ by Lemma 6. Observe that all universal vertices in G_w (that is, $|\overline{C}_i| = 1$) have been disregarded by Lemma 7.
3. Handling connected and co-connected graphs:
   (a) If $G_w = T$ then $A(G_w) =$ the set of leaves in T by Observation 4.
   (b) If $\overline{G_w} = T$ then $A(G_w) =$ the set of non-leaves in T by Lemma 8.
   (c) Otherwise, include G_w in the collection $\mathcal{G}$.

All steps can be carried out in $O(n + m)$ time by checking the type of the internal node w in $T(G)$ and assigning the components and the co-components with the subtrees of w's children. Testing the corresponding cases whenever G_w is connected and co-connected can be done by looking at the number of edges of G_w, that is in time $O(|V(G_w)| + |E(G_w)|)$. Therefore the algorithm outputs in $O(n+m)$ time the described collection $\mathcal{G}$ and the set $A(G) \setminus A(\mathcal{G})$. □

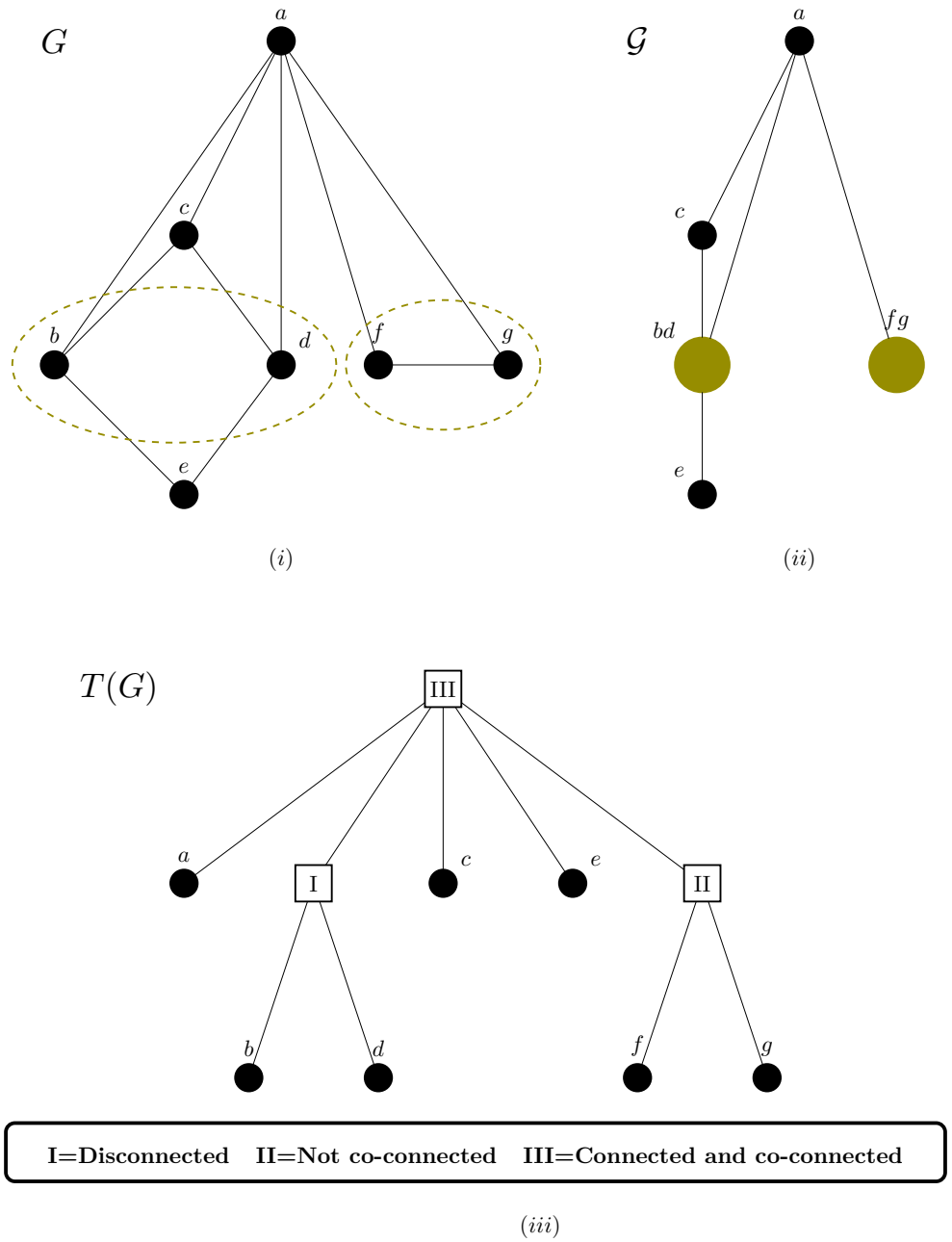

Figure 3.7: Illustrating the modular decomposition tree of a graph G given in (i).

CHAPTER 4

Computing Avoidable Vertices Directly from G

Here we give two different approaches for computing all avoidable vertices of a given graph G. Both of them deal with the input graph itself without shrinking any unnecessary information, as opposed to the algorithms given in forthcoming sections. Our first algorithm makes use of notions related to minimal triangulations of G and runs in time $O(n^2m)$. The second algorithm runs in time $O(n^2 + m^2)$ and is based on a modified, traditional breadth-first search algorithm.

Let us first explain our algorithm through a minimal triangulation of G. We first need some necessary definitions. A graph is *chordal* if it does not contain an induced cycle of length more than three. In different terminology, G is chordal if and only if G is $(C_4, C_5, \ldots)$-free graph.

A graph $H = (V, E \cup F)$ is a *minimal triangulation* of $G = (V, E)$ if H is chordal and for every $F' \subset F$, the graph $(V, E \cup F')$ is not chordal. The edges of F in H are called *fill edges*. Several $O(nm)$-time algorithms exist for computing a minimal triangulation [4, 6, 14, 18]. In connection with avoidable vertices, Beisegel et al. [3] showed the following characterization.

4.1 A nice minimal triangulation

Theorem 10 ([3])**.** *Let u be a vertex of G. Then u is avoidable in G if and only if u is a simplicial vertex in some minimal triangulation of G.*

Although such a characterization is complete, it does not lead to an efficient algorithm for deciding whether a given vertex is avoidable, since one has to produce *all* possible minimal triangulations of G (see Figure 4.1). Here we

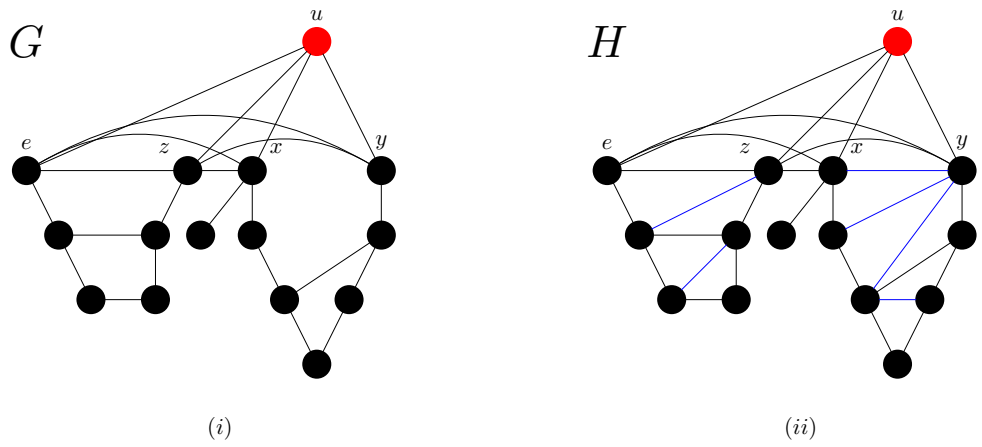

Figure 4.1: An avoidable vertex u in a graph G that is a simplicial vertex in **some** minimal triangulation H of G.

strengthen such a characterization in the sense that it provides an efficient recognition based on a particular, *nice*, minimal triangulation of G.

Lemma 11. *Let u be a vertex of a graph $G = (V, E)$ and let $H = (V, E \cup F)$ be a minimal triangulation of G such that u is not incident to any edge of F. Then u is avoidable in G if and only if u is simplicial in H.*

Proof. If u is simplicial in H then by Theorem 10 we deduce that u is avoidable in G. Suppose that u is non-simplicial in H. Then there are two vertices $x, y \in N_G(u)$ that are non-adjacent in H. Since G is a subgraph of H, we have $xy \notin E(G)$ (see Figure 4.2). We claim that there is no path in G between x and y that avoids any vertex of $N_G[u] \setminus \{x, y\}$. Assume for contradiction that there is such a path P. Then $V(P) \setminus \{x, y\}$ is non-empty and contains vertices only from $V \setminus N[u]$. This means that x, y belong to the same connected component of H induced by $(V \setminus N[u]) \cup \{x, y\}$. As u is non-adjacent to any vertex of $V \setminus N[u]$ in H, the vertices of $(V \setminus N[u]) \cup \{x, y, u\}$ induce an induced cycle of length at least four in H. Then we reach a contradiction to the chordality of H. Therefore, there is no such path between x and y, which implies that u is non-avoidable in G. □

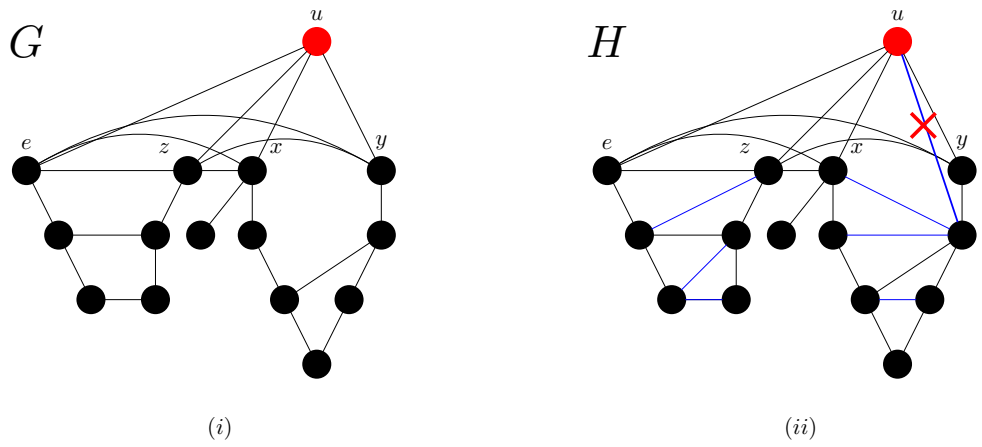

Figure 4.2: An avoidable vertex u of G is a non-simplicial vertex in a minimal triangulation H of G. Notice that an incident edge to u is added as a fill edge.

4.2 Constructing a nice minimal triangulation via vertex incremental approach

Next we show that such a minimal triangulation with respect to u, always exist and can be computed in $O(nm)$ time. Our approach for computing a nice minimal triangulation of G is *vertex incremental*, in the following sense. We take the vertices of G one by one in an arbitrary order $(v_1, \ldots, v_n)$, and at step i we compute a minimal triangulation H_i of $G_i = G[\{v_1, \ldots, v_i\}]$ from a minimal triangulation H_{i-1} of G_{i-1} by adding only edges incident to v_i. This is possible thanks to the following result.

Lemma 12 ([8])**.** *Let G be an arbitrary graph and let H be a minimal triangulation of G. Consider a new graph $G' = G + v$, obtained by adding to G a new vertex v. There is a minimal triangulation H' of G' such that $H' - v = H$.*

We denote by $H(v_1, \ldots, v_n)$ a vertex incremental minimal triangulation of G which is obtained by considering the vertex ordering $(v_1, \ldots, v_n)$ of G. Computing such a minimal triangulation of G, based on any vertex ordering, can be done in $O(nm)$ time [8].

Lemma 13. *Let u be a vertex of G and let $X = N_G(u)$ and $A = V(G) \setminus N_G[u]$. In any vertex incremental minimal triangulation $H(A, u, X)$ of G, no fill edge is incident to u.*

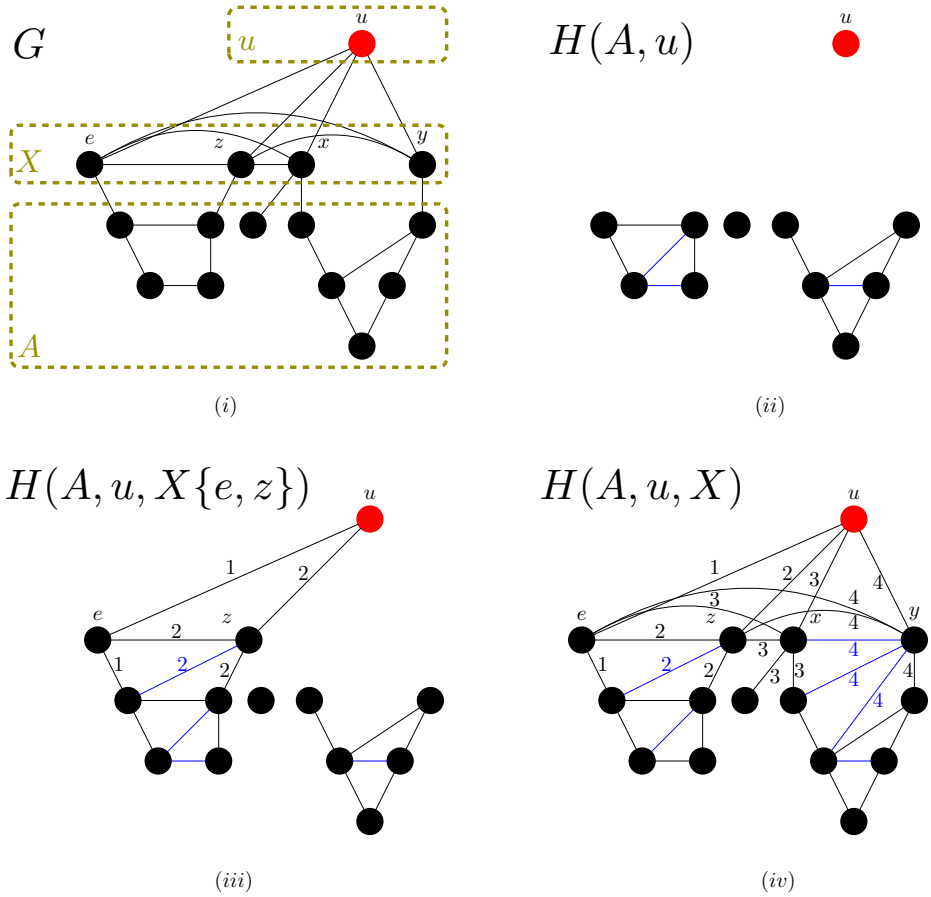

Figure 4.3: Illustrating the construction of a *nice* minimal triangulation via *vertex incremental* approach. (i) In order to check avoidability for an avoidable vertex u in a graph G we consider the vertex sets $\{u\}, X, A$. (ii) Starting the *vertex incremental* by adding in an arbitrary order all the vertices of A and then we add vertex u. Note that u is not adjacent to any vertex of A. (iii) We assume that we have this vertex order (e, z, x, y) for X. First we add vertex e and all edges labeled by 1 in black that already existed in G and then none fill edge is needed to be added. Secondly vertex z and edges labeled 2 in black are added in the same way as for vertex e and additionally fill edges are added, labeled by 2 in blue. (iv) Finally we add, in the exact same way, vertices x and y and thus we end up in a *nice* minimal triangulation in which u is simplicial.

Algorithm 1: Testing if u is avoidable with a vertex incremental minimal triangulation

Input : A graph G, a minimal triangulation H of G, and a vertex u
Output: Returns true iff u is avoidable in G

```
1 Let X = N_G(u) and A = V(G) \ N_G[u];
2 Initialize a new graph H' = H[A ∪ {u}];
3 Add the vertices of X in H' in an arbitrary order and maintain a
    minimal triangulation H' of G by applying the O(nm)-time
    algorithm given in [8];
4 if u is simplicial in H' then
5 |   return true;
6 else
7 |   return false;
```

Proof. Let $H(A, u, X) = (V, E \cup F)$ be a vertex incremental minimal triangulation of $G = (V, E)$. Consider the vertex ordering (A, u, X). Observe that when adding u to $H[A]$ no fill edge is required, as the considered graph $H[A] + u$ is already chordal. Moreover u is adjacent in G to every vertex appearing after u in the described ordering (A, u, X). Thus u is non-adjacent to any vertex of A in $H(A, u, X)$ which means that no edge of F is incident to u (see Figure 4.3). □

4.3 Avoidable vertex via minimal triangulations

A direct consequence of Lemmas 11 and 13 is an $O(nm)$-time recognition algorithm for deciding whether a given vertex u is avoidable. For every vertex u, we first construct a vertex incremental minimal triangulation $H(A, u, X)$ of G by applying the $O(nm)$-time algorithm given in [8]. Then we simply check whether u is simplicial in the chordal graph $H(A, u, X)$ by Lemma 11, which means that the overall running time is $O(nm)$.

We note that one may compute any minimal triangulation H of G, as a preprocessing step in time $O(nm)$, and then use H for constructing the vertex incremental minimal triangulation at each vertex u, so that $H[A]$ is already computed for $A = V(G) \setminus N_G[u]$. Although such an approach results within the same theoretical time complexity, in practice it avoids recomputing common parts of the input data. We give the details in Algorithm 1 and, as already

explained, its running time is $O(nm)$. By applying Algorithm 1 on each vertex, we obtain the following result.

Theorem 14. *Listing all avoidable vertices of G by using Algorithm 1 takes $O(n^2m)$ time.*

An interesting remark of such an approach is that we can list all avoidable vertices of a chordal graph G in an efficient way. We note that such a result can be obtained directly from the definition of an avoidable vertex which shows that a non-simplicial vertex of a chordal graph is non-avoidable.

Corollary 15. *Let G be a chordal graph. Listing all avoidable vertices of G can be done in $O(n^\omega)$ time, where $O(n^\omega)$ is the time required to multiply two $n \times n$ binary matrices.*

Proof. By Lemma 11 the set of simplicial vertices of G is the set of avoidable vertices because any minimal triangulation H of G contains no fill edge, as G is chordal. Thus listing the avoidable vertices of a chordal graph G reduces to listing the simplicial vertices of G. Therefore detecting all avoidable vertices can be done in $O(n^\omega)$ time by using the algorithm of [15], which is the time needed to perform a fast matrix multiplication. □

4.4 A fast algorithm for listing avoidable vertices

Our second approach is based on the following notion of *protecting* that we introduce here. Given a set of vertices $S \subseteq V$, an S-excluded path is a path in which no internal vertex belongs to S. Observe that an edge is an S-excluded path, for any choice of S. By definition a single vertex is connected to itself by the trivial path. Whenever there is an S-excluded path in G between vertices a and b, notice that a can reach b through vertices of $V(G) \setminus S$ (see Figure 4.4).

Definition 2 (protecting)**.** Let x and y be two vertices of G. We say that x *protects* y if there is a $N_G[y]$-excluded path between x and every vertex of $N_G(y)$. In other words, x protects y if for any $z \in N_G(y) \setminus \{x\}$, either $xz \in E(G)$ or x can reach z through vertices of $V(G) \setminus N_G[y]$ (see Figure 4.5).

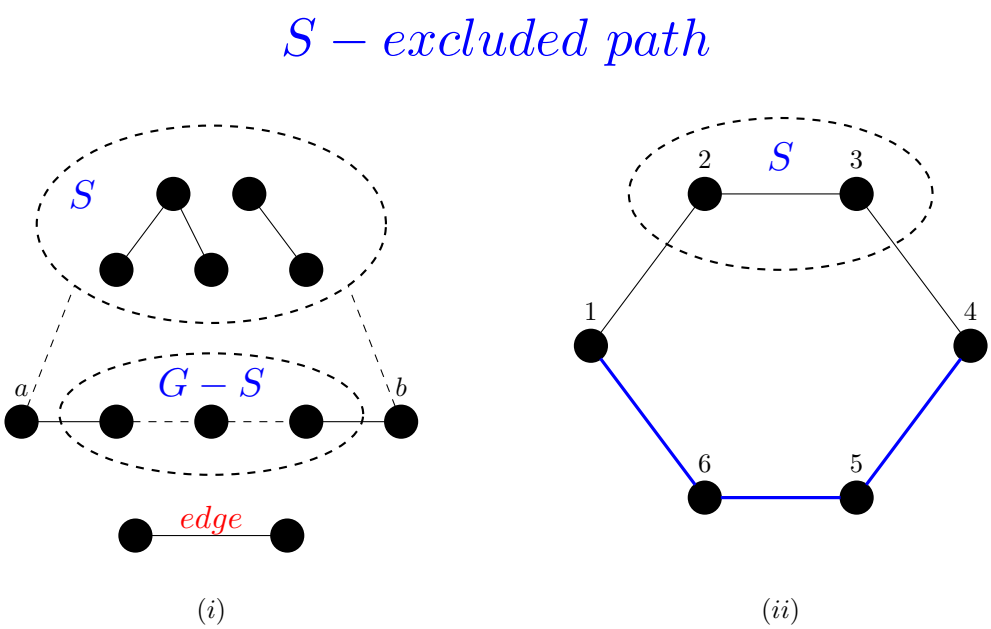

Figure 4.4: (i) A path $\langle a, b\rangle$ is a $S-$*excluded path* if none of its internal vertices belong to set S. Note that an edge is a $S-$*excluded path* for any choice of S. (ii) An example for $S-$*excluded path*. Observe that for the vertices 1 and 4 there is a $(\{2\},\{3\})-$*excluded path* but not a $(\{2\},\{5\})-$*excluded path*.

4.4.1 A modified BFS

Let us explain how to check if x protects y in linear time, that is in $O(n+m)$ time. We consider the graph $G' = G - \{y\}$ and run a slight modification of a breadth-first search algorithm on G' starting from x. In particular, we try to reach the vertices of $N_G(y) \setminus \{x\}$ (target set) from x in G'. Every time we encounter a vertex v of the target set, we include v in a set T of discovered target vertices and we do not continue the search from v by avoiding to place v within the search queue. Consequently, no vertex of the target set is a non-leaf node of the constructed search tree. Algorithm 2 shows in detail the considered modification of a breadth-first search (see Figure 4.6).

Lemma 16. *Algorithm 2 is correct and runs in $O(n+m)$ time.*

Proof. For the correctness, let T be the search tree discovered by the algorithm

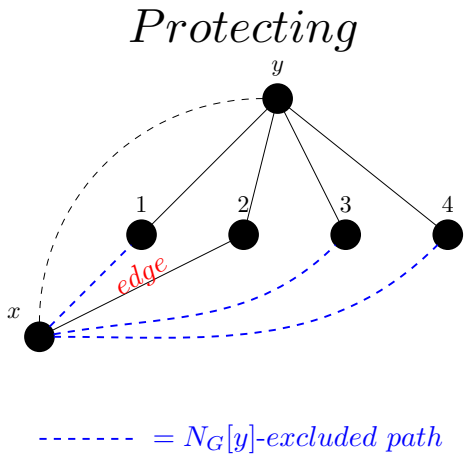

Figure 4.5: For two vertices x, y of a graph G we say that x is protecting y if for x and every vertex $z \in N_G(y)$ there is a $N_G[y] - excluded\,path$.

when the search starts from x. Observe that the basic concepts of the breadth-first search are maintained, so that the key properties with the shortest paths between the vertices of G and the search tree T are preserved. If there is a leaf vertex v in the constructed tree T such that $v \in S$ then the unique path in T is an S-excluded path in G between x and v, since no vertex of S is a non-leaf vertex of T. On the other hand, assume that there is an S-excluded path in G between x and every vertex of S. For every $v \in S$, among such S-excluded paths between x and v, choose $P(v)$ to be the shortest. Let $p(v)$ be the neighbor of v in $P(v)$. Clearly x and every vertex $p(v)$ belong to the same connected component of G. Consider the graph $G-S$. Notice that every vertex $p(v)$ belongs to the same connected component with x in $G - S$, since for otherwise some vertices of S separate x and a vertex v of S which implies that there is no S-excluded path in G between x and v in G. Now let T_x be a breadth-first search tree of $G - S$ that contains x. Then the distance between x and $p(v)$ in T_x corresponds to the length of their shortest path in $G - S$. Construct T by attaching every vertex v of S to be a neighbor of $p(v)$ in T_x. Therefore T is a tree that contains the shortest S-excluded paths between x and the vertices of S.

Regarding the running time, notice that no additional data structure is required compared to the classical implementation of the breadth-first search. Hence the running time of Algorithm 2 is bounded by the breadth-first search algorithm which is $O(n + m)$. □

Algorithm 2: Detecting whether there is an S-excluded path between x and every vertex of S

Input : A graph G, a vertex x, and a target set $S \subseteq V(G)$
Output: Returns true iff there is an S-excluded path between x and every vertex of S

```
1  Set Q = {x}, T = ∅;
2  Mark x;
3  while Q is not empty do
4      s = Q.pop();
5      for v ∈ N(s) do
6          if v is unmarked then
7              if v ∈ S then
8                  T = T ∪ {v};
9              else
10                 Q.add(v);
11             Mark v;
12 return T = S
```

4.4.2 Exploiting the modified BFS

Therefore we can check whether x protects y by running Algorithm 2 on the graph $G - \{y\}$ with target set $S = N_G(y) \setminus \{x\}$. The connection to the avoidability of a vertex, can be seen with the following result.

Lemma 17. *Let u be a vertex of a graph $G = (V, E)$. Then u is avoidable in G if and only if x protects u for every vertex $x \in N_G(u)$.*

Proof. Suppose first that u is avoidable. Consider a vertex $x \in N_G(u)$. Then for any vertex $y \in N_G(u) \setminus \{x\}$ there is a path between x and y that avoids vertices of $N_G(u)$. This means that there is an S-excluded path between x and y with $S = N_G[u]$. Thus x protects u in G.

For the other direction, assume that u is non-avoidable. Then there are vertices $x, y \in N_G(u)$ that belong to different connected components of $G - (N_G[u] \setminus \{x, y\})$. Thus x cannot reach y through vertices of $V(G) \setminus N_G[u]$, implying that x (and y) does not protect u. Therefore there are at least two vertices in $N_G(u)$ that do not protect u. □

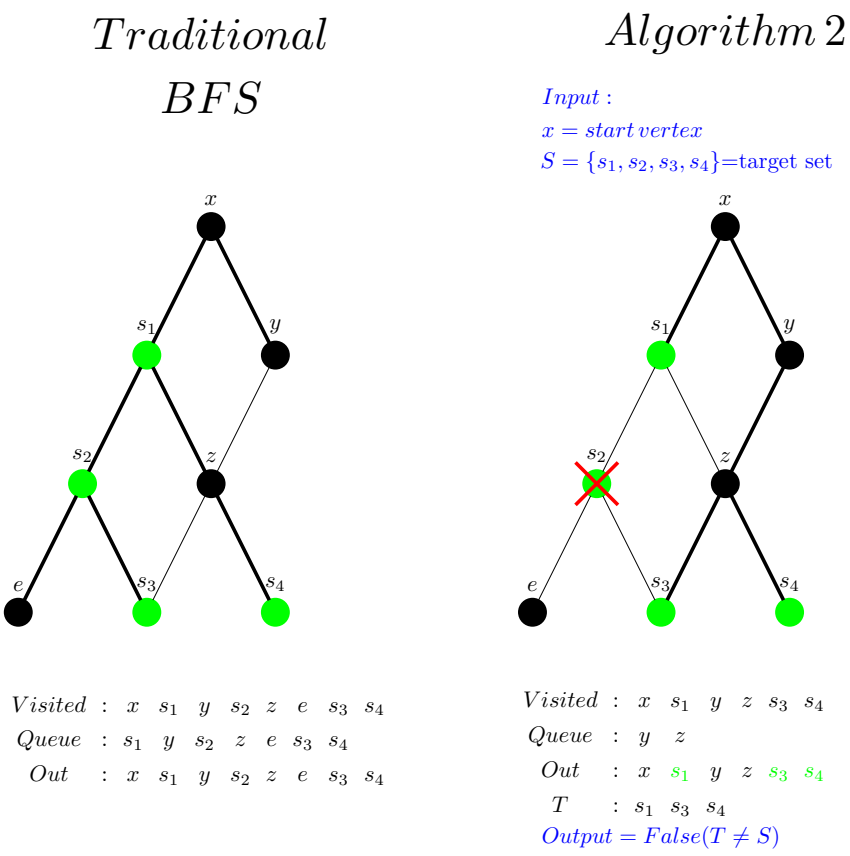

Figure 4.6: Illustrating how Algorithm 2 works taking advantage of the modified BFS, in comparison to traditional BFS. We assume that start vertex is x and the target set $S = \{s_1, s_2, s_3, s_4\}$.

Now we are ready to show our fast algorithm for deciding whether a vertex is avoidable which is given in Algorithm 3.

Theorem 18. *Listing all avoidable vertices of G by using Algorithm 3 takes $O(n^2 + m^2)$ time.*

Proof. Correctness follows from Lemmas 16 and 17. For the running time, observe that constructing G' takes $O(n+m)$ time. Moreover we need to make $d(u)$ calls to Algorithm 2 for a particular vertex u where $d(u)$ is the degree of u. Thus, by Lemma 16 the total running time is $O(\sum_u (1 + d(u))(n + m)) = O(n^2 + m^2)$. □

Algorithm 3: Testing if u is avoidable by detecting whether its neighbors protect u

Input : A graph G and a vertex u
Output: Returns true iff u is avoidable in G

```
1 Let X = N_u and G' = G − {u};
2 for x ∈ X do
3 |   Set S = X \ {x};
4 |   if Algorithm 2(G', x, S) is not true then
5 |   |   return false;
6 return true;
```

CHAPTER 5

Avoidable Vertices via Contractions

Here we show how to compute all avoidable vertices of a graph G through contractions. Given a graph $G = (V_G, E_G)$ and a vertex $u \in V_G$, we denote by G_u the graph obtained from G by contacting every connected component of $G - N_G[u]$. We partition the vertices of $G_u - u$ into (X, C), such that $X = N_G(u)$ and C contains the contracted vertices of $G - N_G[u]$. We denote by $G_u(X, C)$ the contracted graph where (X, C) is the vertex partition with respect to G_u. Observe that $G_u[X \cup \{u\}] = G[X \cup \{u\}]$ and $G_u[C \cup \{u\}]$ is an independent set (see Figure 5.1).

5.1 Computing a contracted graph

Observation 19. *Given a vertex u of $G = (V, E)$, the construction of $G_u(X, C)$ can be done in $O(n + m)$ time.*

Proof. To compute the connected components $C_1, \ldots, C_k$ of $G - N_G[u]$ takes linear time. For each vertex set C_i, $1 \leq i \leq k$, we compute $N_G(C_i)$ in time $d(C_i)$ where $d(C_i)$ is the sum of the degrees of the vertices in C_i. As $C_1, \ldots, C_k$ is a partition of $V(G) \setminus N_G[u]$, the total running time for substituting each set C_i is $O(k + \sum d(C_i)) = O(n + m)$. □

Next we show that $G_u(X, C)$ holds all necessary information of important paths of G with respect to the avoidability of u.

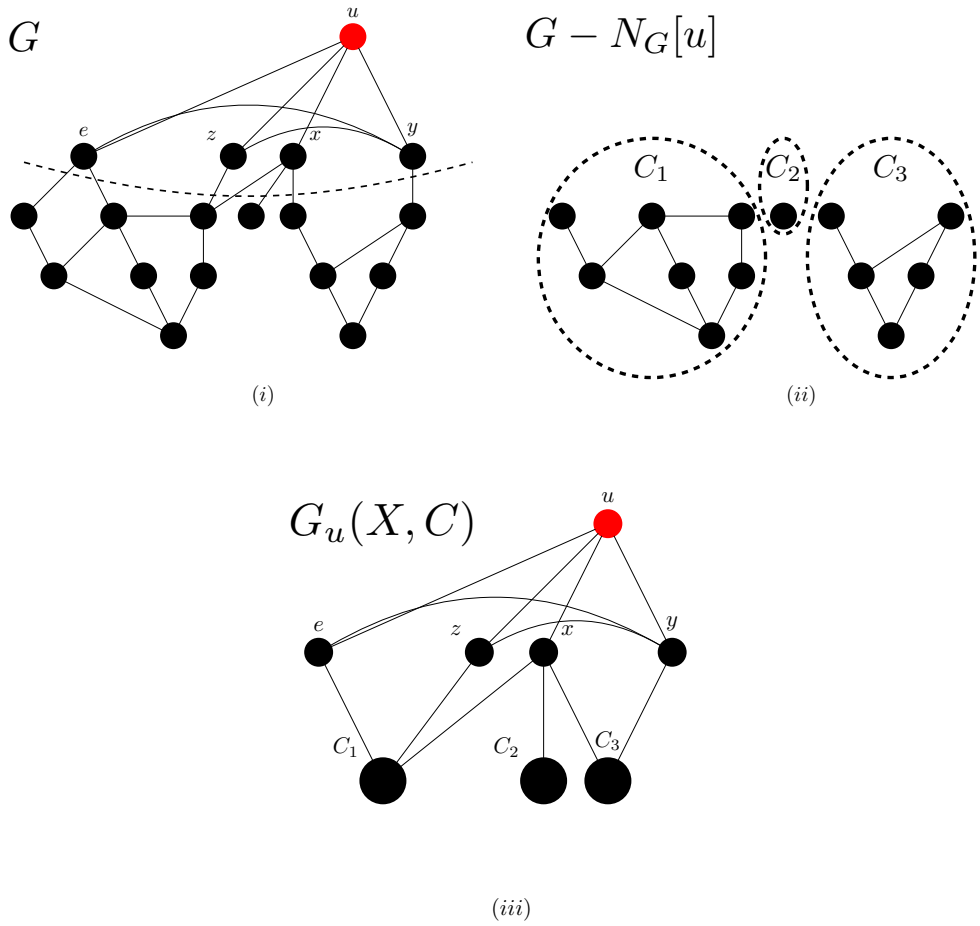

Figure 5.1: (i) An avoidable vertex u of a graph G. (ii) After removing $N_G[u]$, we find the connected components in $O(n+m)$ time. (iii) Illustrating $G_u(X, C)$, by contracting the connected components of (ii).

5.2 Avoidable vertex in a contracted graph

Lemma 20. *Let u be a vertex of a graph $G = (V, E)$. Then u is avoidable in G if and only if u is avoidable in $G_u(X, C)$.*

Proof. Since $G[X \cup \{u\}] = G_u[X \cup \{u\}]$, we only need to consider the vertices of $X = N_G(u)$ that are non-adjacent. Let $x, y \in N_G(u) = X$ such that $xy \notin E(G)$ and let $S = (X \cup \{u\}) \setminus \{x, y\}$. Observe that all vertices of S belong

to both graphs G and $G_u(X, C)$. We claim that there is a path between x and y in $G - S$ if and only if there is a path between x and y in $G_u(X, C) - S$. Consider any path in $G - S$ of the form $\langle x, P, y\rangle$. The vertices of the given path belong to the same connected component of $G - S$. Thus the vertices of P belong to exactly one connected component C_P of $G - (S \cup \{x, y\})$. As $S \cup \{x, y\} = N_G[u]$, there is a vertex $C_i \in C$ that corresponds to C_P in the contracted graph $G_u(X, C)$. Hence, the path $\langle x, C_i, y\rangle$ forms the desired path in $G_u(X, C) - S$.

If there is a path between x and y in $G_u(X, C) - S$ then such a path is of length two and has the form $\langle x, C_i, y\rangle$ where $C_i \in C$. Since xC_i is an edge in $G_u(X, C)$, there is a vertex $a \in V(C_i)$ such that $xa \in E(G)$. Similarly, there is a vertex $b \in V(C_i)$ such that $yb \in E(G)$. As a and b belong to the same connected component C_i of $G - N_G[u]$, there is a path P_i in G between a and b that contains only vertices from $V(C_i)$. Thus there is a path $\langle x, P_i, y\rangle$ in G where $P_i \subseteq V(C_i)$.

Now observe that any path between two neighbors of u in either $G - S$ or $G_u(X, C) - S$ does not contain any vertex of $N_G[u]$. Therefore, by the above claim, we get the desired characterization of u in both graphs. □

Lemma 20 implies that we can apply all of our algorithms given in the previous section in order to recognize an avoidable vertex. Although such an approach does not lead to faster theoretical time bounds, in practice the contracted graph has substantial smaller size than the original graph and may lead to practical running times. We next show that the contracted graph results in an additional algorithm with different running time. Let $G_u(X, C)$ be the contracted graph of a vertex u. The *filled-contracted graph*, denoted by $H_u(X, C)$, is the graph obtained from $G_u(X, C)$ by adding all necessary edges in order to make clique every neighborhood of $C_i \in C$. That is, for every $C_i \in C$, $N_{H_u}(C_i)$ is a clique (see Figure 5.2). The following proof resembles the characterization given through minimal triangulations in Lemma 11. However observe that $H_u(X, C)$ is not necessarily a chordal graph, because $X \nsubseteq N_{G_u}(C)$.

Lemma 21. *A vertex u is avoidable in G if and only if $H_u[X]$ is a clique.*

Proof. We apply Lemma 20 and we need to show that u is avoidable in $G_u(X, C)$ if and only if $H_u[X]$ is a clique. Assume that u is avoidable in $G_u(X, C)$. We show that $H_u[X]$ is a clique. Consider two vertices $x, y \in X$. If xy is an edge in $G_u(X, C)$ then xy remains an edge in $H_u(X, C)$, as $G_u(X, C)$

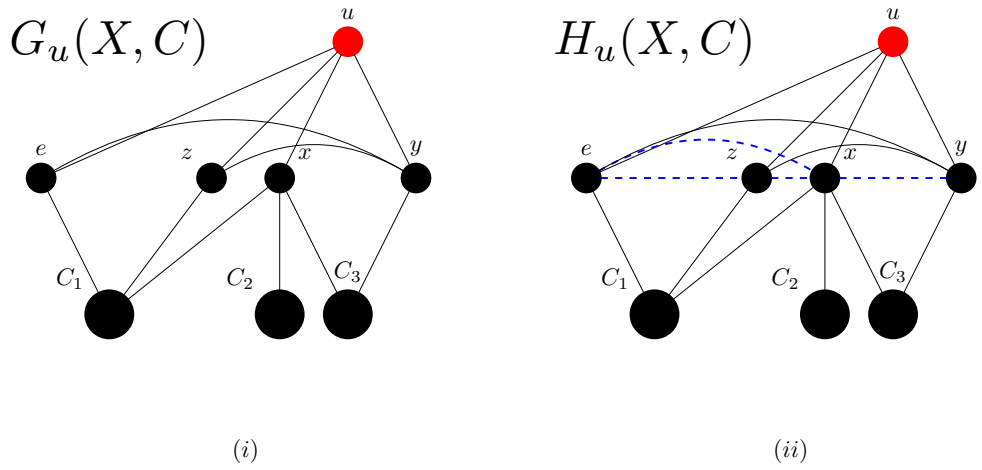

Figure 5.2: Illustrating how the filled-contracted graph $H_u(X, C)$ is obtained from $G_u(X, C)$. Observe that u is simplicial in $H_u(X, C)$.

is a subgraph of $H_u(X, C)$. If x and y are non-adjacent in $G_u(X, C)$, there is a vertex $C_i \in C$ such that $\{x, y\} \subseteq N_{G_u}(C_i)$, because u is avoidable and $G_u[C]$ is an independent set. Thus, by the definition of $H_u(X, C)$, $N_{H_u}(C_i)$ is a clique implying that xy is an edge in $H_u[X]$.

Assume that u is non-avoidable in $G_u(X, C)$. Then there are vertices $x, y \in X$ such that $xy \notin E(G_u)$ and they belong in different connected components of $G_u[C \cup \{x, y\}]$. Thus x and y is a pair of non-adjacent vertices in $H_u[X]$, since there is no vertex $C_i \in C$ such that $x, y \in N_{G_u}(C_i)$. Hence there is a pair of non-adjacent vertices in $H_u[X]$, so that $H_u[X]$ is not a clique. □

5.3 Exploiting a fast matrix multiplication

We take advantage of Lemma 21 in order to recognize whether u is avoidable. The naive construction of $H_u(X, C)$ requires $O(n^3)$ time, since $|X| \leq n$ and $|C| \leq n$. Instead of constructing $H_u(X, C)$, we are able to check $H_u[X]$ in an efficient way through matrix multiplication. To do so, we consider the graph G' obtained from $G_u(X, C)$ by removing u and deleting every edge with both endpoints in X. Observe that the resulting graph G' is a bipartite graph with bipartition (X, C), as $G_u[C \cup \{u\}]$ is an independent set. It turns out that it

Algorithm 4: Testing if u is avoidable by using matrix multiplication

Input : A graph G and a vertex u
Output: Returns true iff u is avoidable in G

```
1  Construct the contracted graph G_u(X, C) of u;
2  Let G_1 = G_u(X, C) − {u};
3  Construct the adjacency matrix M_1 of G_1;
4  Let G_2 be the bipartite graph obtained from G_1 by removing every
     edge having both endpoints in X;
5  Construct the adjacency matrix M_2 of G_2;
6  Compute the square of M_2, i.e., M_2^2 = M_2 · M_2;
7  Construct the matrix M_3 = M_1 + M_2^2;
8  for x, y ∈ X do
9      if the entry M_3[x, y] is zero then
10         return false;
11 return true;
```

is enough to check whether two vertices of X are in distance two in G' which can be encapsulated by the square of its adjacency matrix. Algorithm 4 shows in details our proposed approach.

We are now in position to claim the following running time through matrix multiplication.

Theorem 22. *Listing all avoidable vertices of G by using Algorithm 4 takes $O(n^{1+\omega})$ time, where $O(n^{\omega})$ is the time required to multiply two $n \times n$ binary matrices.*

Proof. We apply Algorithm 4 on each vertex of G. Let us first discuss on the correctness of Algorithm 4. By Lemma 21, it is enough to show that $H_u[X]$ is a clique if and only if $M_3[X]$ has non-zero entries in its non-diagonal positions. Let G_1 and G_2 be the two constructed graphs in Algorithm 4. Observe that the square of G_2, denoted by G_2^2, is the graph obtained from the same vertex set of G_2 and two vertices u, v are adjacent in G_2^2 if the distance of u and v is at most two in G_2. Thus the matrix M_2^2 computed by Algorithm 4 corresponds to the adjacency matrix of G_2^2. Now it is enough to notice that two vertices x, y of X are adjacent in $H_u[X]$ if and only if $xy \in E(G_1) \cup E(G_2^2)$. In particular observe that if x and y have a common neighbor w in G_2 then w is a vertex of C since there is no edge between vertices of X in G_2 and $u \notin V(G_2)$. Therefore $M_3[x, y]$ has a non-zero entry if and only if x and y are adjacent in $H_u[X]$.

Regarding the running time, notice that the construction of G_u take linear time by Observation 19. All steps besides the computation of M_2^2 can be done in $O(n^2)$ time. The most time-consuming step is the matrix multiplication involved in computing M_2^2, which can be done in $O(n^\omega)$ time. Hence the total running time for recognizing all n vertices takes $O(n^{1+\omega})$ time. □

CHAPTER 6

Recognizing Avoidable Edges and Paths

Natural generalizations of avoidable vertices are avoidable edges and avoidable paths. Here we show how to efficiently recognize an avoidable edge and an avoidable path. Recall that the two vertices having degree one in an induced path P_k on $k \geq 2$ vertices are called *endpoints*. Moreover, the edge obtained after removing the endpoints from an induced path P_4 on four vertices is called *middle edge*.

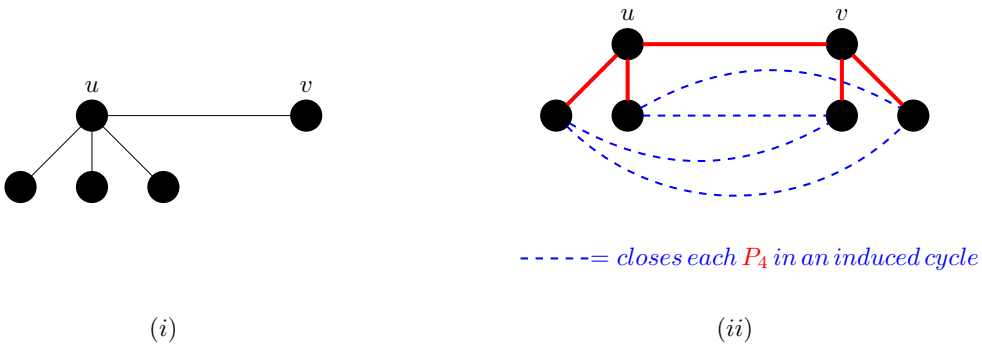

Figure 6.1: (i) The edge uv is simplicial and thus avoidable. (ii) Illustrating an avoidable edge uv.

Definition 3 (simplicial and avoidable edge)**.** An edge uv is called *simplicial* if there is no P_4 having uv as a middle edge. An edge uv is called *avoidable* if either uv is simplicial, or every P_4 with middle edge uv is contained in an induced cycle (see Figure 6.1).

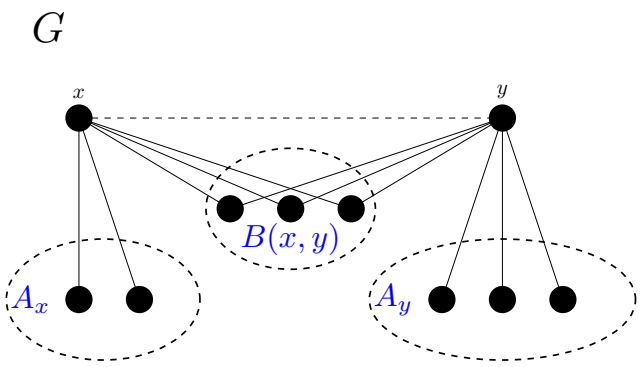

Figure 6.2: Illustrating the sets $B(x,y)$, A_x and A_y for the vertices x, y of a graph G.

6.1 Private and common neighbors

Given two vertices x and y of G, we define the following sets of the neighbors of x and y:

- $B(x,y)$ contains the common neighbors of x and y; i.e., $B(x,y) = N_G(x) \cap N_G(y)$.
- A_x contains the private neighbors of x; i.e., $A_x = N_G(x) \setminus (B(x,y) \cup \{y\})$.
- A_y contains the private neighbors of y; i.e., $A_y = N_G(y) \setminus (B(x,y) \cup \{x\})$.

Under this terminology, observe that $A_x \cap A_y = \emptyset$ and $N_G(\{x,y\})$ is partitioned into the three sets $B(x,y), A_x, A_y$ (see Figure 6.2). Clearly all described sets can be computed in $O(d(x) + d(y))$ time.

Observation 23. *An edge xy of G is simplicial if and only if $A_x = \emptyset$ or $A_y = \emptyset$ or every vertex of A_x is adjacent to every vertex of A_y.*

Proof. Consider a $P_4 = a, x, y, b$ that contains xy as a middle edge. Then $a \in A_x$ and $b \in A_y$ because $ay \notin E(G)$ and $xb \notin E(G)$. Thus both sets A_x and A_y are non-empty. Moreover, since $ab \notin E(G)$, we deduce that any non-edge with one endpoint in A_x and the other in A_y results in a P_4 having xy as a middle edge. □

By Observation 23, the recognition of a simplicial edge can be achieved in $O(n+m)$ time: consider the bipartite subgraph $H(A_x, A_y)$ of $G[A_x \cup A_y]$ which

is obtained by removing every edge having both endpoints in either A_x or A_y. Then it is enough to check whether $H(A_x, A_y)$ is a complete bipartite graph.

We show that the more general concept of an avoidable edge can be recognized in $O(nm)$ time. For doing so, we will take advantage of Algorithm 2 and the notion of protecting given in Definition 2.

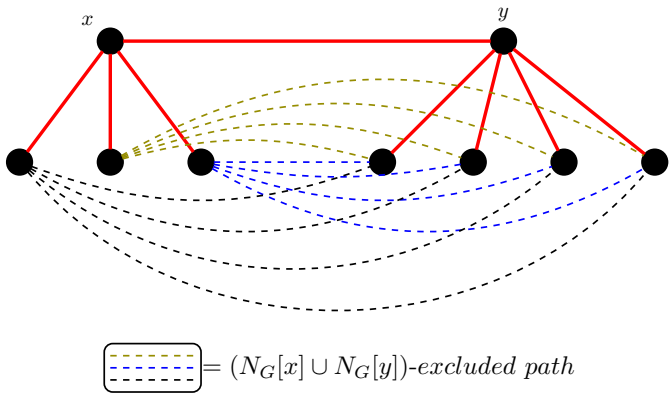

Figure 6.3: Illustrating a protected edge xy.

6.2 Recognizing an avoidable edge

Definition 4 (protected edge)**.** An edge xy is *protected* if there is an $(N_G[x] \cup N_G[y])$-excluded path between every vertex of $N_G(x)$ and every vertex of $N_G(y)$ (see Figure 6.3).

We note that if an edge xy is protected then x protects y and y protects x in accordance to Definition 2. However, the reverse is not necessarily true, as shown in Figure 6.4.

Lemma 24. *Let xy be an edge of G. Then xy is an avoidable edge in G if and only if xy is a protected edge in $G - B(x, y)$.*

Proof. Let $H = G - B(x, y)$ and let us first show that xy is an avoidable edge in G if and only if xy is an avoidable edge in H. Suppose that xy is an avoidable edge in G. For any two vertices $a \in A_x$ and $b \in A_y$ such that $ab \notin E(G)$, there is an induced cycle C that contains a, x, y, b. Now observe that no vertex of $B(x, y)$ belongs to C, as C is an induced cycle in G. Thus

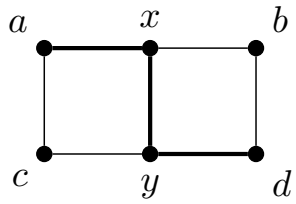

Figure 6.4: In this example we have $N_G[x] \cup N_G[y] = V(G)$. Observe that x protects y, because x has $\{c, y, d\}$-excluded paths to both c and d, and similarly y protects x. However, the edge xy is not protected because, for instance, there is no $V(G)$-excluded path (and, thus, an edge) between a and d. Also notice that there is a $P_4 = \langle a, x, y, d\rangle$ that is not contained in an induced cycle.

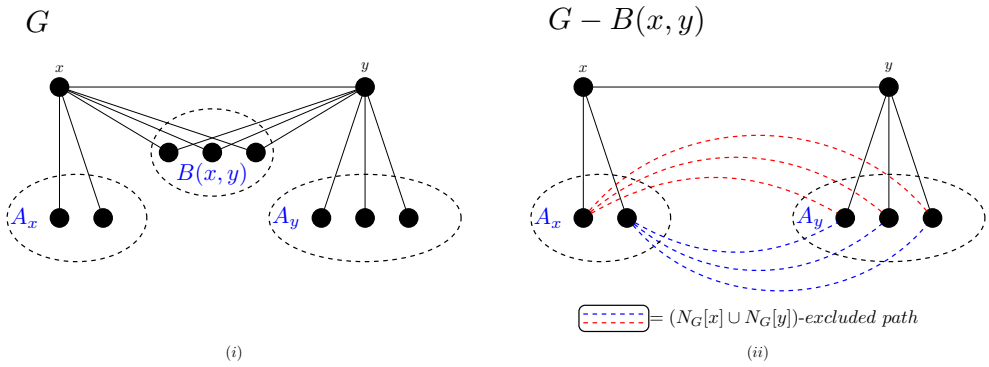

Figure 6.5: Illustrating Lemma 24.

xy is an avoidable edge in H. For the converse, notice that H is an induced subgraph of G, so that all induced cycles of H remain induced cycles in G. Therefore our task is to show that xy is an avoidable edge in H if and only if xy is protected in the same graph H.

Suppose that xy is an avoidable edge in H. Observe that $N_H(x) = A_x \cup \{y\}$ and $N_H(y) = A_y \cup \{x\}$. If at least one of A_x, A_y is empty then xy is protected (as well as simplicial), because all required $(N_H[x] \cup N_H[y])$-excluded paths have length one between a vertex and its neighbors. Consider any two vertices $a \in A_x$ and $b \in A_y$. Clearly the edges xa and yb constitute $N_H[y]$-excluded path and $N_H[x]$-excluded path, respectively. Assume first that $ab \notin E(H)$. Then there is a $P_4 = \langle a, x, y, b\rangle$ that contains xy as a middle edge. Any induced

cycle C that contains the described P_4, contains vertices from $V(H)\setminus(A_x \cup A_y)$, so that the vertices of $C - P_4$ belong to $V(H) \setminus (N_H[x] \cup N_H[y])$. Thus the subpath on C taken from $C - P_4$ with endpoints a and b is a $(A_x \cup A_y \cup \{x, y\})$-excluded path of length at least two between a and b. If $ab \in E(H)$ then $\langle a, b\rangle$ is an $(A_x \cup A_y \cup \{x, y\})$-excluded path of length one between a and b. In all cases we deduce that xy is a protected edge.

Suppose that xy is a protected edge in H. Consider a $P_4 = \langle a, x, y, b\rangle$ that contains xy as middle edge. Then clearly $a \in A_x$, $b \in A_y$, and $ab \notin E(H)$. We show that there is an induced cycle in H that contains the P_4. Between a and b, there is an $(N_H[x] \cup N_H[y])$-excluded path P_{ab} in H. The length of P_{ab} is at least two, since $ab \notin E(H)$. By definition, all internal vertices of P_{ab} belong to $V(H) \setminus (N_H[x] \cup N_H[y])$ and, thus, are non-adjacent to x and y. Let $S = V(P_{ab})$ and consider the induced subgraph $H[S]$ that is connected. Then the shortest path P'_{ab} between a and b in $H[S]$ is an induced path of H. Therefore the concatenation of the $P_4 = \langle a, x, y, b\rangle$ with P'_{ab} results in the desired induced cycle of H. □

Based on Lemma 24, we deduce the following running time for recognizing an avoidable edge. This is achieved by carefully applying Algorithm 2. Notice that the stated running time is comparable to the $O(d(u)(n + m))$-time algorithm for recognizing an avoidable vertex u implied by Theorem 18.

Theorem 25. *Recognizing an avoidable edge of a graph G can be done in $O(n \cdot m)$ time.*

Proof. Let xy be an edge of G. We first collect the vertices of $B(x, y)$ in $O(n)$ time. By Lemma 24 we need to check whether xy is protected in $H = G - B(x, y)$(see Figure 6.5). If xy is simplicial edge then xy is avoidable and, by Observation 23, this can be tested in $O(n + m)$ time. Otherwise, both sets A_x, A_y are non-empty. Without loss of generality, assume that $|A_x| \leq |A_y|$. In order to check if xy is protected, we run $|A_x|$ times Algorithm 2:

- for every vertex $a \in A_x$, run Algorithm 2$(H-((A_x\setminus\{a\})\cup\{x, y\}), a, A_y)$.

In particular, we test whether there is an A_y-excluded path between a and every vertex of A_y without considering the vertices of $(A_x \setminus \{a\}) \cup \{x, y\}$, that is on the graph $H - ((A_x \setminus \{a\}) \cup \{x, y\})$. If all vertices of A_x have an A_y-excluded path with all the vertices of A_y on each corresponding graph, then such paths do not contain any internal vertex from $A_x \cup A_y \cup \{y\}$. Since

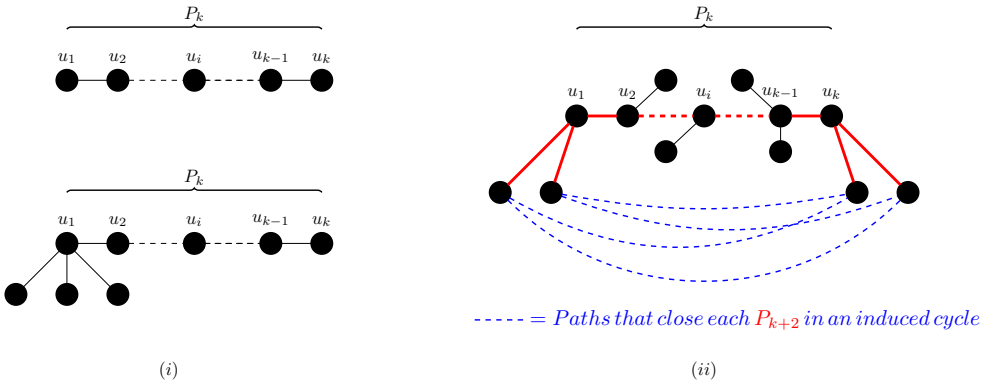

Figure 6.6: (i) A path P_k that is simplicial and thus avoidable. (ii) A non-simplicial avoidable path P_k.

$N_H[x] = A_x \cup \{x, y\}$ and $N_H[y] = A_y \cup \{x, y\}$, we deduce that xy is a protected edge, and thus, xy is avoidable in G. Regarding the running time, observe that we make at most $n \geq |A_x|$ calls to Algorithm 2 on induced subgraphs of G. Therefore, by Lemma 16, the total running time is $O(nm)$. □

6.3 Recognizing an avoidable path

Let us now show how to extend the recognition of an avoidable edge towards their common generalization of avoidable induced paths. The *internal path* of a non-edgeless induced path P is the path obtained from P without its endpoints and it is denoted by $in(P)$.

Definition 5 (simplicial and avoidable path)**.** An induced path P_k on $k \geq 2$ vertices is called *simplicial* if there is no induced path on $k + 2$ vertices that contains P_k as an internal path. An induced path P_k on $k \geq 2$ vertices is called *avoidable* if either P_k is simplicial, or every induced path on $k + 2$ vertices that contains P_k as an internal path is contained in an induced cycle (see Figure 6.6).

For $k = 2$, avoidable paths correspond to avoidable edges. Let P_k be an induced path on k vertices of a graph G with $k \geq 3$ having endpoints x and y. We denote by $I[P_k]$ the vertices of $N_G[in(P_k)] \setminus \{x, y\}$. That is, $I[P_k]$ contains the vertices of the internal path of P_k and their neighbors outside P_k.

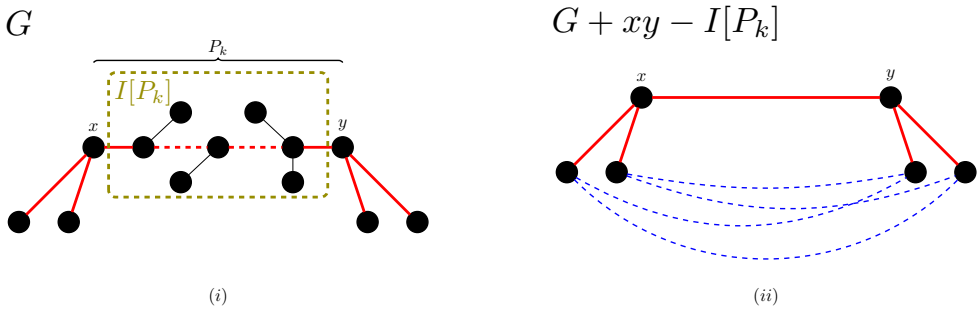

Figure 6.7: Illustrating Lemma 26 by replacing P_k by an edge.

Given two non-adjacent vertices x and y in G, we denote by $G + xy$ the graph obtained from G by adding the edge xy (see figure 6.7).

Lemma 26. *Let P_k be an induced path on k vertices of a graph G with $k \geq 3$ having endpoints x and y. Then P_k is an avoidable path in G if and only if xy is an avoidable edge in $G + xy - I[P_k]$.*

Proof. We claim first that there is a P_{k+2} that contains P_k as an internal path in G if and only if there is a P_4 that contains xy as a middle edge in the graph $H = G + xy - I[P_k]$.

Assume that there is a P_{k+2} that contains P_k as an internal path in G. Let x' and y' be the endpoints of P_{k+2}. As P_{k+2} is an induced path, both x', y' belong to H and $x'y, xy', x'y' \notin E(H)$. Thus $\langle x', x, y, y' \rangle$ is a P_4 in H that contains xy as a middle edge.

Assume that there is a $P_4 = \langle x', x, y, y' \rangle$ in H that contains xy as a middle edge. Consider the vertices of the path P_{k-2} of $P_k - \{x, y\}$ in G that correspond to the edge xy of H. Then no vertex of the P_{k-2} is adjacent to any of x' or y' by the construction of H. Thus, replacing the edge xy in the $P_4 = \langle x', x, y, y' \rangle$ by the path P_{k-2}, results in an induced path P_{k+2} on $k + 2$ vertices in G.

Observe that the above claim implies that P_k is a simplicial path in G if and only if xy is a simplicial edge in H. Next we show that a non-simplicial path P_k with endpoints x and y is avoidable in G if and only if the non-simplicial edge xy is avoidable in H. Assume that there is a $P_{k+2} = \langle x', x, P_{k-2}, y, y' \rangle$ that contains $P_k = \langle x, P_{k-2}, y \rangle$ as an internal path in G. Let C_G be an induced cycle that contains the P_{k+2} in G. Since C_G is induced cycle, every vertex of $C_G - P_{k-2}$ belongs to H. Now observe that the vertices of $C_G - P_{k-2}$ induce

a path in G of length at least four. Hence the vertices of $C_G - P_{k-2}$ induce a cycle in H, since $xy \in E(H)$, which shows that xy is avoidable edge in H.

To show that P_k is avoidable in G, we show that there is an induced cycle that contains the described P_{k+2}. Let C_H be an induced cycle of H containing a $P_4 = \langle x', x, y, y' \rangle$. Since xy is a avoidable edge in H, such a cycle exists. Construct the cycle C' obtained from C_H by removing the edge xy and attaching the path P_{k-2} of $P_k - \{x, y\}$. Then C' is an induced cycle in G because:

- $C_H - \{x, y\}$ is an induced path in G, as $H - \{x, y\}$ is an induced subgraph of G,
- P_k is an induced path in G by definition, and
- no vertex of P_{k-2} has a neighbor in $C_H - \{x, y\}$, as $N_G(P_{k-2}) \setminus \{x, y\} \subset I[P_k]$.

Therefore there is an induced cycle in G that contains the described P_{k+2} of P_k. □

Theorem 27. *Given an induced path P_k on $k > 2$ vertices of G, testing whether P_k is avoidable can be done in $O(n \cdot m)$ time.*

Proof. Assume that the endpoints of P_k are x and y. By Lemma 26, it is enough to check if the edge xy is avoidable in the graph $G + xy - I[P_k]$. Constructing the graph $G + xy - I[P_k]$ takes $O(nk)$ time. Applying the algorithm given in Theorem 25 results in an algorithm with the claimed running time, since $k \leq n$. □

CHAPTER 7

Empirical Analysis

Here we conduct an experimental analysis of the proposed algorithms for computing all avoidable vertices of a graph. We mainly compare, in term of time execution, four algorithms of the previous chapters. We also propose two additional algorithms that we present in this chapter. All considered six algorithms compute avoidable vertices of a graph that is both connected and co-connected.

We wrote our implementations in Python, using NetworkX, a python package for studying data structure and network connection analysis to compile the code. Also we used Matplotlib package for visualizing and plotting the graph regarding the avoidability of its vertices. These two packages use a pretty huge amount of resources such as CPU and memory consumption, so it is only for study and analysis purpose.

We report the running times on a Dell Precision Tower 7820 CTO Base machine running Ubuntu (16.04 LTS), equipped with an Intel Xeon Gold 5118 2.3 GHz processor with 16 MB L3 cache and 192GB DDR4-2400 RAM at 2,666 MHz. We did not use any parallelization, and each algorithm ran on a single core. All our running times were averaged over ten different runs.

7.1 Experimental algorithms

We have implemented the naive algorithm that we denote by NG. NG is based on the definition of an avoidable vertex. Given a vertex u, the naive algorithm NG runs a linear-time connectivity algorithm (here we considered BFS) for every pair (x, y) of non-adjacent vertices of $N_G(u)$ that is performed on the graph $G - (N_G[u] \setminus \{x, y\})$. As already explained, the running time of NG is $O(n^3m)$.

Algorithm 5: Testing if $H_u[X]$ is a clique.

Input : A graph G and a vertex u
Output: Returns true iff u is avoidable in G

```
1 Construct the contracted graph G_u(X, C) of u;
2 Construct the filled-contracted graph H_u(X, C) of u;
3 if H_u[X] is a clique then
4 |   return true;
5 else
6 |   return false;
```

Moreover, we use two of the aforementioned algorithms, by applying the notion of *protecting* (Definition 2 and Lemma 17) via Algorithm 3 which is based on the modified BFS. The reason for reaching two such algorithms is that of directly applying to the whole graph as well as on the contracted graph by Lemma 20. We denote the two algorithms based on the protecting notion by PG and PCG, respectively. Regarding their running time, Theorem 18 shows that they are both performed in $O(n^2 + m^2)$ time.

Let us now explain three additional algorithms that are applied on the contracted graph $G_u(X, C)$ of a vertex u.

We propose an algorithm denoted by HuCG, that comes directly from Lemma 21, in which it is stated that a vertex u of G is avoidable, if and only if, the filled-contracted graph $H_u[X]$ is a clique. Algorithm 5 gives the details of the proposed algorithm. Regarding its running time, observe that constructing $H_u(X, C)$ takes $O(n^3)$ time, as we need to consider all pairs of neighbors of a vertex $C_i \in C$ in order to turn $N_{H_u}(C_i)$ into a clique.

Furthermore, we propose the ICG algorithm that takes advantage of the intersection of the neighborhood of two non-adjacent vertices in X.

Lemma 28. *A vertex u is avoidable in G if and only if for every pair of non-adjacent vertices $(x, y) \in X$, there is a $C_i \in C$ such that $C_i \in (N_{G_u}(x) \cap N_{G_u)}(y))$.*

Proof. By Lemma 21, u is avoidable if and only if $H_u[X]$ is a clique. If there are two non-adjacent vertices x and y that have no common neighbor in C then xy is not an edge of H_u. Thus $H_u[X]$ is not a clique and u is a non-avoidable vertex. On the other hand, if every pair of non-adjacent vertices $x, y \in X$ have a common neighbor in C then $H_u[X]$ is a clique which means

Algorithm 6: Testing if each non-adjacent pair $x, y \in N_G(u)$ have a common neighbor in C.

Input : A graph G and a vertex u
Output: Returns true iff u is avoidable in G

```
1 Construct the contracted graph G_u(X, C) of u;
2 Let L={(x, y) : (x, y) ∉ E(G) and x, y ∈ X};
3 Remove all edges of X and remove u;
4 for (x, y) ∈ L do
5     if N_{G_u}(x) ∩ N_{G_u}(y) = ∅ then
6         return false;
7 return true;
```

that u is avoidable. □

In other words, a vertex u is avoidable if and only if every pair of non-adjacent vertices of $N_{G_u}(u)$ have a common neighbor in C. Algorithm 6 shows the details of ICG. Regarding its running time, observe that the intersection of $N_{G_u}(x) \cap C$ and $N_{G_u}(y) \cap C$ takes $O(n)$ time and as there $O(n^2)$ pairs (x, y) in X, the overall running time for testing a single vertex u is $O(n^3)$.

Further, we present the CXCG algorithm that uses the complement F_u of a subgraph of $G_u(X, C)$. Let us explain. We consider $G_u(X, C)$, we remove vertex u and all edges between vertices of X. At the new bipartite graph F with partition (X, C), we take its *bipartite complement*, regarding only the edges between X and C. The resulting graph is denoted by F_u (see Figure 7.2). In particular, $V(F_u) = X \cup C$ and $E(F_u) = \{\{x, c\} | x \in X, c \in C, \{x, c\} \notin E(G_u)\}$. With the next lemma, we claim that u is non-avoidable iff there is at least one pair (x, y) of non-adjacent vertices of X such that $N_{F_u}(x) \cup N_{F_u}(y) = C$. That is, u is non-avoidable iff there are at most two non-adjacent vertices in X that *dominate* the set C in F_u.

Lemma 29. *Let u be a vertex of G and let F_u be the bipartite complement of the contracted graph $G_u(X, C)$. Then u is a non-avoidable vertex in G if and only if there is a pair of non-adjacent vertices $(x, y) \in X$ such that $N_{F_u}(x) \cup N_{F_u}(y) = C$.*

Proof. Assume that u is non-avoidable. Then from Lemma 28 there are non-adjacent vertices $x, y \in X$ that have no common neighbor in C. We show that x and y dominate C in F_u. Let $N_x = N_{G_u}(x) \cap C$ and let $N_y = N_{G_u}(y) \cap C$.

Algorithm 7: Testing if there is a non-adjacent pair $(x, y) \in N_G(u)$, such that y is adjacent to every node C_i of $F_u - N_{F_u}[x]$.

Input : A graph G and a vertex u
Output: Returns true iff u is avoidable in G

```
1 Construct the contracted graph G_u(X, C) of u;
2 Let L={(x, y) : (x, y) ∉ E(G) and x, y ∈ X};
3 Construct the bipartite complement graph F_u;
4 for (x, y) ∈ L do
5 |   remove N_{F_u}[x];
6 |   if N_{F_u}(y) = C then
7 |   |   return false;
8 return true;
```

Then C is partitioned into three disjoint sets $C \setminus (N_x \cup N_y), N_x, N_y$. In the bipartite complement F_u, x is adjacent to every vertex of $C \setminus (N_x \cup N_y), N_y$ and y is adjacent to every vertex of $C \setminus (N_x \cup N_y), N_x$. Thus every vertex of C is dominated by x or y, which implies that $N_{F_u}(x) \cup N_{F_u}(y) = C$.

Now assume that there is a vertex $C_i \in C$ such that $C_i \notin (N_{F_u}(x) \cup N_{F_u}(y))$, for every pair of non-adjacent vertices $x, y \in X$. Then C_i is adjacent to both x and y in G_u, since C_i is a vertex of C. Thus x and y have a common neighbor in C in the graph G_u. By Lemma 28 we deduce that u is avoidable. □

In order to take advantage of Lemma 29 and test if there exist two non-adjacent vertices x and y in X, we first construct the bipartite complement F_u of $G_u(X, C)$, then we remove all vertices of $N_{F_u(x)}$, and then we check whether y is adjacent to every vertex of C (see Figure 7.1). The details of the aforementioned CXCG algorithm are given in Algorithm 7. Regarding its running time, the construction of the bipartite complement F_u takes $O(n^2)$ time and testing each of the $O(n^2)$ pair of vertices takes $O(d_{F_u}(x))$ time where $d_{F_u}(x)$ is the degree of x in F_u. Thus the overall running time of CXCG is $O(n^3\ \overline{m})$.

In Table 7.1 we summarize all considered algorithms that we have implemented and used in this experimental analysis.

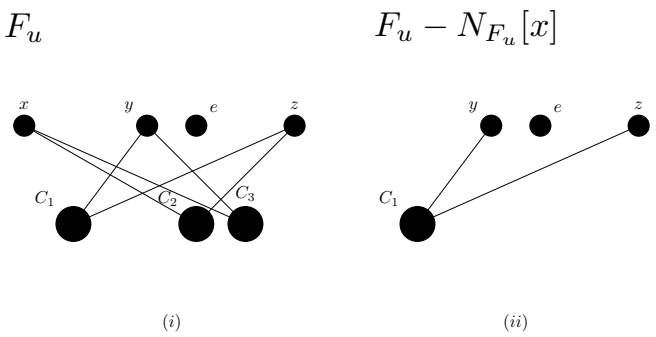

Figure 7.1: Illustrating CXCG algorithm in F_u graph. Observe that after the removal of vertex x and its neigbours in C, the remaining vertex y is adjacent to all of the remaining vertices of C.

Algorithm	**Description**	**Complexity**
NG	Applying the naive algorithm directly in G	$O(n^3m)$
PG	Using the notion of protecting directly in G (Algorithm 3)	$O(n^2 + m^2)$
PCG	Using the notion of protecting in the contracted G (Algorithm 3)	$O(n^2 + m^2)$
HuCG	Checking the filled-contracted graph $H_u[X]$ in the contracted G (Algorithm 5)	$O(n^4)$
ICG	Checking the intersection of the neighborhood in the contracted G (Algorithm 6)	$O(n^4)$
CXCG	Testing for a dominating pair of non-adjacent vertices of $G_u(X)$ in F_u in the contracted G (Algorithm 7)	$O(n^3\overline{m})$

Table 7.1: The algorithms that are used in the experimental analysis. Here n, m and $\overline{m}$ denote the number of vertices, edges, and non-edges, respectively, of the given graph.

7.2 Random graphs

Here we compare the efficiency of our algorithms in random graphs. In order to do so, we used random graph generator of NetworkX package with 100, 200 and 300 nodes. Moreover for each random graph we used three different values of probability p, for edge creation : p=0.05, p=0.50 and p=0.95 (see table 7.3).

Table 7.2 shows the considered random graphs in which we report structural properties in each generated graph. Besides the usual data, we report the percentage of the number of non-avoidable vertices and the reduced sizes (in

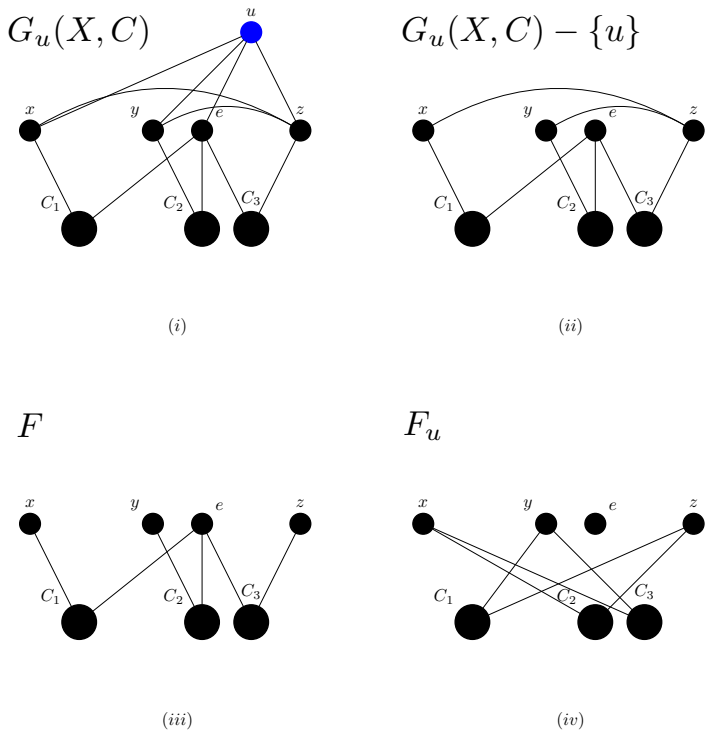

Figure 7.2: Illustrating the steps of Lemma 29.

average) of the contracted graphs.

The results in running times for all of our algorithms applied on the graphs taken from Table 7.2 are given in Table 7.3.

7.3 Real-world graphs

For the experimental evaluation of our algorithms in real-world networks, we use real datasets that are both connected and co-connected. The datasets are taken from several divergent areas such as animal networks, brain networks, social networks, infrastructure networks and facebook networks [20].

Table 7.4 reports the details of each considered real-world network. We have chosen 12 networks in which the number of vertices ranges from 58 to 1458 and with edge density (i.e., $2m/n(n-1)$) that diverges significantly.

n	p	m	Density	Aver. Deg.	Non-Avoid. (%)	C.R. (%)
100	0.05	337	0.068	6.74	2	6
100	0.50	2475	0.500	49.50	0	29
100	0.95	4613	0.931	92.26	2	89
200	0.05	1175	0.059	11.75	2	2
200	0.50	9950	0.500	99.50	0	27
200	0.95	18725	0.940	187.25	0	90
300	0.05	2512	0.056	16.74	3	2
300	0.50	22425	0.500	149.50	0	26
300	0.95	42338	0.943	282.25	0	90

Table 7.2: Random graphs produced by NetworkX and their percentage of non-avoidable vertices.The percentage of C.R. shows how much each graph is reduced in terms of number of vertices, after the contraction procedure.

n	p	NG	PG	PCG	HuCG	ICG	CXCG
100	0.05	0.5745	0.4939	0.0743	**0.0737**	0.0772	0.0786
100	0.50	18.2944	7.7577	0.4048	**0.2798**	0.8779	0.5677
100	0.95	18.8785	5.2704	1.4625	**0.7490**	5.1182	0.8820
200	0.05	7.1858	4.8089	0.3514	**0.3480**	0.3751	0.3835
200	0.50	278.7692	109.1687	2.8438	**1.8785**	8.8196	4.9547
200	0.95	242.1613	55.7412	11.2460	**5.3009**	66.9655	7.0046
300	0.05	30.8796	18.0353	0.8825	**0.8717**	0.9484	0.9631
300	0.50	1493.5639	525.0821	9.5477	**6.1045**	38.3495	19.2981
300	0.95	1147.9665	243.6962	39.2457	**18.0800**	309.1732	24.9562

Table 7.3: Running times of our algorithms, in random graphs produced by NetworkX. We highlight with bold the best running time in each instance (row).

The results in running times for all of our algorithms applied on the graphs taken from Table 7.4 are given in Table 7.5.

7.4 Experimental conclusions

The corresponding plotted values from Table 7.3 and Table 7.5 are shown in Figure 7.3 and Figure 7.5 respectively.

For the random graphs that are neither sparse nor dense, ($0.05 < p < 0.95$) HuCG has the best performance. The same behavior occurs on sparse random graphs ($p = 0.05$) as well as on dense random graphs ($p = 0.95$). However, on real-world graphs in almost all instances both PCG and HuCG outperform

Dataset	n	m	Density	Aver. Deg.	Non-Avoid. (%)	C.R. (%)
insecta-ant-colony2-day39	58	879	0.531	30.310	22	45
soc-dolphins	62	159	0.084	5.129	23	13
insecta-ant-colony2-day16	111	4041	0.661	72.810	36	60
aves-wildbird-network-1	131	1444	0.169	22.045	65	23
insecta-ant-colony6-day01	164	10731	0.802	130.865	16	71
inf-USAir97	332	2126	0.038	12.807	35	17
bio-celegans	453	2025	0.019	8.940	35	6
bio-diseasome	516	1188	0.008	4.604	31	4
soc-wiki-Vote	889	2914	0.007	6.555	23	3
socfb-Reed98	962	18812	0.040	39.110	13	3
socfb-Haverford76	1446	59589	0.057	82.419	5	3
bio-yeast	1458	1948	0.001	2.672	31	2

Table 7.4: The datasets from Real World Networks taken from [20] that we used for the evaluation of our algorithms.

Dataset	NG	PG	PCG	HuCG	ICG	CXCG
insecta-ant-colony2-day39	1.5365	0.7510	0.1117	**0.0756**	0.1685	0.0987
soc-dolphins	0.0691	0.0851	**0.0249**	0.0250	0.0251	0.0258
insecta-ant-colony2-day16	16.1304	6.3688	0.8177	**0.5784**	2.0461	0.7049
aves-wildbird-network-1	0.6090	0.5732	**0.1952**	0.2058	0.2050	0.2002
insecta-ant-colony6-day01	107.1668	38.1257	3.9208	**2.2945**	17.0078	3.6033
inf-USAir97	2.8236	2.7650	**0.5876**	0.6102	0.6157	0.6816
bio-celegans	3.9009	5.0335	**1.0322**	1.0532	1.0329	1.0621
bio-diseasome	**0.2920**	0.3045	0.4345	0.4382	0.4381	0.4377
soc-wiki-Vote	17.2691	19.4446	**3.4021**	3.4156	3.4144	3.4541
socfb-Reed98	1582.0383	597.3261	15.3947	**15.1578**	18.2741	18.0137
socfb-Haverford76	20 416.9665	6229.3409	70.4331	**66.8618**	105.9508	95.9866
bio-yeast	4.5956	7.6304	4.3437	**4.3279**	4.3423	4.3497

Table 7.5: Results in running times for our algorithms regarding real world network datasets. We highlight with bold the best running time in each instance (row).

the rest of the algorithms.

The results taken from random graphs in Table 7.3 suggest that the contraction process significantly reduces the actual running time. When comparing PC and PCG which apply the same algorithm, an improvement of almost 85% is occurred. As expected, the small number of non-avoidable vertices is crucial for the exhaustive application of each algorithm. Moreover, sparsity and density of each test graph is an important aspect that affects the corresponding running times. Furthermore, Table 7.5 shows that the outcomes taken from the random graphs naturally correspond on real-world data.

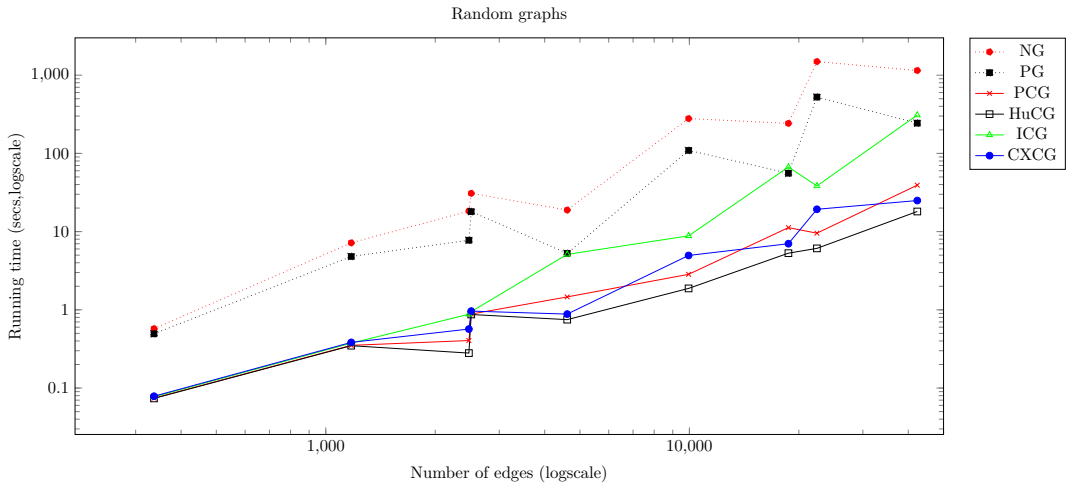

Figure 7.3: Running times in seconds with respect to the number of edges (in log scale) taken by Table 7.3.

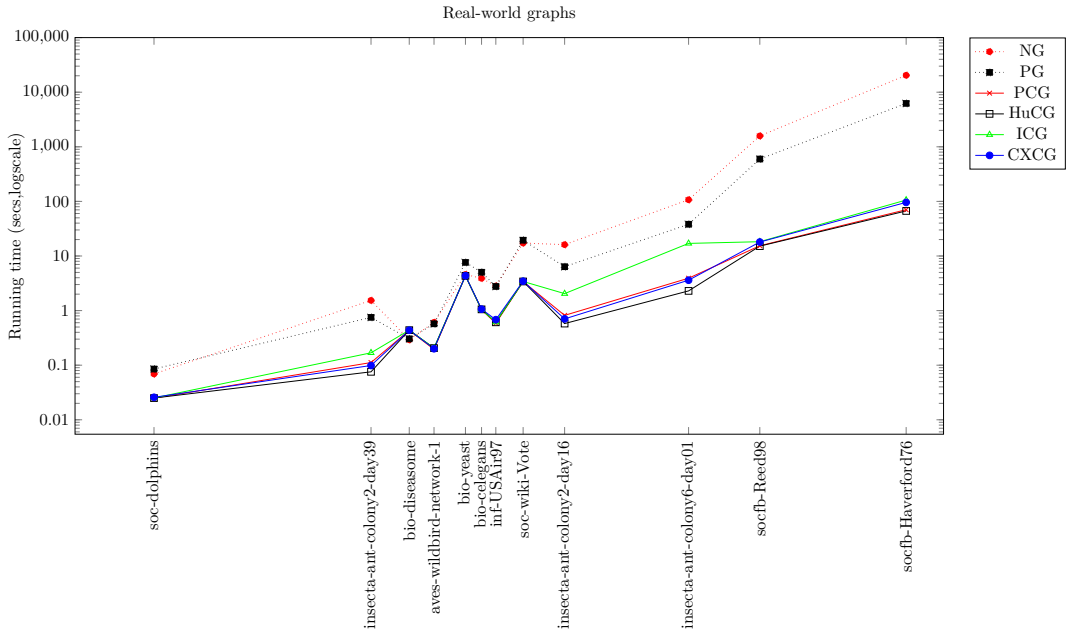

Figure 7.4: Running times in seconds with respect to the number of edges (in log scale) taken by Table 7.5.

CHAPTER 8

Concluding Remarks

The running times of our algorithms for listing all avoidable vertices are comparable to the corresponding ones for listing all simplicial vertices. Thus we believe it is difficult to achieve a reduction of the running time for avoidable vertices without affecting the time needed for simplicial vertices. As pointed out, we can detect avoidable vertices in particular graph classes in more efficient way. Towards this direction, it is interesting to consider planar graphs and reveal any possible improvement on the running time. Moreover the notion of protecting and the relative S-excluded paths seem to tackle further problems concerning avoidable structures. Our recognition algorithm for avoidable edges results in an algorithm for listing avoidable edges with running time $O(nm^2)$ which is comparable to the $O(m^2)$-algorithm for listing avoidable vertices. Regarding avoidable paths on k vertices, one needs to detect first with a naive algorithm a path P_k in $O(n^k)$ time and then test whether P_k being avoidable or not. As observed in [9], such a detection is nearly optimal, since we can hardly avoid the dependence of the exponent in $O(n^k)$. Therefore by Theorem 25 we get an $O(n^{k+1} \cdot m)$-algorithm for listing all avoidable paths on k vertices. An interesting direction for further research along the avoidable paths is to reveal problems that can be solved efficiently by taking advantage the list of all avoidable paths in a graph.

Bibliography

[1] Pierre Aboulker, Pierre Charbit, Nicolas Trotignon, and Kristina Vuskovic. Vertex elimination orderings for hereditary graph classes. *Discret. Math.*, 338(5):825–834, 2015.

[2] Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 522–539. SIAM, 2021.

[3] Jesse Beisegel, Maria Chudnovsky, Vladimir Gurvich, Martin Milanic, and Mary Servatius. Avoidable vertices and edges in graphs. In Zachary Friggstad, Jörg-Rüdiger Sack, and Mohammad R. Salavatipour, editors, *Proceedings of WADS 2019*, volume 11646, pages 126–139, 2019.

[4] Anne Berry. A wide-range efficient algorithm for minimal triangulation. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 860–861. ACM/SIAM, 1999.

[5] Anne Berry, Jean R. S. Blair, Jean Paul Bordat, and Geneviève Simonet. Graph extremities defined by search algorithms. *Algorithms*, 3(2):100–124, 2010.

[6] Anne Berry, Jean R. S. Blair, Pinar Heggernes, and Barry W. Peyton. Maximum cardinality search for computing minimal triangulations of graphs. *Algorithmica*, 39(4):287–298, 2004.

[7] Anne Berry and Jean Paul Bordat. Separability generalizes dirac's theorem. *Discret. Appl. Math.*, 84(1-3):43–53, 1998.

[8] Anne Berry, Pinar Heggernes, and Yngve Villanger. A vertex incremental approach for maintaining chordality. *Discret. Math.*, 306(3):318–336, 2006.

[9] Marthe Bonamy, Oscar Defrain, Meike Hatzel, and Jocelyn Thiebaut. Avoidable paths in graphs. *Electron. J. Comb.*, 27(4):P4.46, 2020.

[10] J. A. Bondy and U. S. R. Murty. *Graph Theory*. Springer, 2008.

[11] Derek G. Corneil, H. Lerchs, and L. Stewart Burlingham. Complement reducible graphs. *Discret. Appl. Math.*, 3(3):163–174, 1981.

[12] Derek G. Corneil, Yehoshua Perl, and Lorna K. Stewart. A linear recognition algorithm for cographs. *SIAM J. Comput.*, 14(4):926–934, 1985.

[13] G. A. Dirac. On rigid circuit graphs. *Abhandlungen aus dem Mathematischen Seminar der Universitat Hamburg*, 25(1):71–76, 1961.

[14] Pinar Heggernes. Minimal triangulations of graphs: A survey. *Discret. Math.*, 306(3):297–317, 2006.

[15] Ton Kloks, Dieter Kratsch, and Haiko Müller. Finding and counting small induced subgraphs efficiently. *Inf. Process. Lett.*, 74(3-4):115–121, 2000.

[16] R. M. McConnell and J. P. Spinrad. Modular decomposition and transitive orientation. *Discrete Mathematics*, 201:189–241, 1999.

[17] Tatsuo Ohtsuki, Lap Kit Cheung, and Toshio Fujisawa. Minimal triangulation of a graph and optimal pivoting order in a sparse matrix. *Journal of Mathematical Analysis and Applications*, 54(3):622–633, 1976.

[18] Donald J. Rose, Robert Endre Tarjan, and George S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, 5(2):266–283, 1976.

[19] Marc Tedder, Derek G. Corneil, Michel Habib, and Christophe Paul. Simpler linear-time modular decomposition via recursive factorizing permutations. In *Proceedings of ICALP 2008*, volume 5125 of *Lecture Notes in Computer Science*, pages 634–645, 2008.

[20] Ryan A. Rossi and Nesreen K. Ahmed. The Network Data Repository with Interactive Graph Analytics and Visualization. In *Proceedings of AAAI 2015*.