



AVERTISSEMENT

Ce document est le fruit d'un long travail approuvé par le jury de soutenance et mis à disposition de l'ensemble de la communauté universitaire élargie.

Il est soumis à la propriété intellectuelle de l'auteur. Ceci implique une obligation de citation et de référencement lors de l'utilisation de ce document.

D'autre part, toute contrefaçon, plagiat, reproduction illicite encourt une poursuite pénale.

Contact : ddoc-theses-contact@univ-lorraine.fr

LIENS

Code de la Propriété Intellectuelle. articles L 122. 4

Code de la Propriété Intellectuelle. articles L 335.2- L 335.10

http://www.cfcopies.com/V2/leg/leg_droi.php

<http://www.culture.gouv.fr/culture/infos-pratiques/droits/protection.htm>



Output-sensitive algorithms for enumeration problems in graphs

P.H.D. THESIS

presented and publicly supported on December 10th 2019

for obtaining the degree of

Philosophiae Doctor in Informatic

by

Sarah Blind

Directed by Nadia Creignou and Laurent Vigneron

Co-supervised by Kolja Knauer, Frédéric Olive and Petru Valicov

Composition of the jury

Referees: Arnaud Durand, Professor, Université Paris 7
Roberto Grossi, Professor, Università di Pisa

Examiner: Mathieu Liedloff, Associate professor, Université d'Orléans

Supervisors: Nadia Creignou, Professor, Aix-Marseille université
Laurent Vigneron, Professor, Université de Lorraine

Mis en page avec la classe thesul.

Abstract

This thesis is a study, from an algorithmic point of view, of the complexity of solving some enumeration problems in graphs. Algorithms for enumeration problems are meant to produce all solutions of a combinatorial problem, without repetition. Because there is a potentially exponential number of solutions to be generated, different approaches to analyse the performance of those algorithms have been developed: the input-sensitive approach and the output-sensitive approach. The first is a measure of the complexity that is based on the size of the input, and which consists in analysing the total time needed to enumerate the objects. The second is a measure based on the size of the input and the output. Here we will be interested in an output-sensitive approach and we will pay special attention to the notion of delay, i.e., the time needed between the production of two consecutive solutions.

The thesis is divided into two independent parts. In the first one we propose a general framework that allows for the study of enumeration of vertex set properties in graphs. We prove that when such a property is locally definable with respect to some order on the set of vertices, then it can be enumerated with linear delay. Our method is a reduction of the considered enumeration problem to the enumeration of paths in directed acyclic graphs. We then apply this general method to enumerate with linear delay minimal connected dominating sets and maximal irredundant sets in interval graphs and in permutation graphs, as well as maximal irredundant sets in circular-arc graphs and in circular-permutation graphs.

The second part of the thesis is dedicated to the study of k -arc-connected orientations. These are orientations for which at least k arcs have to be removed in order to destroy the strong connectivity. We first give a simple algorithm to enumerate the k -arc-connected orientations respecting some fix outdegree sequence, i.e., a sequence constituted of the outdegrees for each vertex of the input graph. Such orientations are called α -orientations. We then give a simple algorithm to enumerate the outdegree sequences attained by k -arc-connected orientations. Combining both yields an algorithm that enumerates all k -arc-connected orientations of a graph in delay $O(knm^2)$ and amortized time $O(m^2)$, where n , m denote the number of vertices and edges, respectively. Our amortized time improves over another approach using submodular flows and moreover is much simpler, since it is basically a combination of BFS searches.

Résumé

Cette thèse est une étude, d'un point de vue algorithmique, de la complexité de la résolution de certains problèmes d'énumération dans les graphes. Les algorithmes pour les problèmes d'énumération ont pour but la production de toutes les solutions d'un problème combinatoire, et ce sans répétition. Comme il existe un nombre potentiellement exponentiel de solutions à générer, différentes approches pour analyser la performance de ces algorithmes ont été développées: l'approche "input-sensitive" et l'approche "output-sensitive". La première est une mesure de la complexité basée sur la taille de l'entrée, elle consiste en l'analyse du temps total nécessaire à l'énumération des objets. La seconde est une mesure basée sur la taille de l'entrée et de la sortie. Nous nous intéresserons ici à l'approche output-sensitive et nous accorderons une attention particulière à la notion de délai, i.e., le temps nécessaire entre la production de deux solutions consécutives.

La thèse est divisée en deux parties indépendantes. Dans la première, nous proposons un cadre général qui permet d'étudier l'énumération d'ensembles de sommets d'un graphe respectant une certaine propriété. Nous prouvons que lorsqu'une telle propriété peut être définie localement par rapport à un certain ordre sur l'ensemble des sommets du graphe, alors les ensembles de sommets respectant cette propriété peuvent être énumérés avec un délai linéaire. Notre méthode consiste à réduire le problème d'énumération considéré à l'énumération de chemins dans les graphes acycliques dirigés. Nous appliquons ensuite cette méthode générale pour énumérer avec un délai linéaire les ensembles dominants connexes minimaux et les ensembles irredondants maximaux dans les graphes d'intervalles et dans les graphes de permutation, ainsi que les ensembles irredondants maximaux dans les graphes arc-circulaires et dans les graphes permutation-circulaires.

La deuxième partie de la thèse est consacrée à l'étude des orientations k -arc-connexes. Il s'agit d'orientations pour lesquelles il est nécessaire de supprimer au moins k arcs pour détruire la forte connexité. Nous présentons d'abord un algorithme simple pour énumérer les orientations k -arc-connexes respectant une séquence de degrés sortants, i.e., une séquence constituée des degrés sortants pour chaque sommet du graphe pris en entrée. De telles orientations sont appelées α -orientations. Ensuite nous présentons un algorithme simple pour énumérer les séquences de degrés sortants pour lesquelles il existe une orientation k -arc-connexe. En combinant ces deux algorithmes, nous obtenons un algorithme qui énumère toutes les orientations k -arc-connexes d'un graphe avec un délai de $O(knm^2)$ et un temps amorti de $O(m^2)$, où n et m dénotent le nombre de sommets et d'arêtes du graphe, respectivement. Notre temps amorti améliore une approche antérieure basée sur les flots sous-modulaires ; il est en outre beaucoup plus simple puisqu'il s'agit d'une combinaison de recherches BFS.

Contents

1		
Introduction		
1.1	Enumeration: algorithms and complexity	1
1.2	Contributions of the thesis	3
1.3	Structure and content of the thesis	5
2		
Complexity for enumeration		
2.1	Complexity classes for enumeration	7
2.2	Methods for efficient enumeration	11
2.2.1	Flashlight Backtrack Search	11
2.2.2	Reverse Search	12
2.2.3	Reductions	15
2.3	Hardness results	16
3		
Efficient enumeration of vertex set properties in graphs		
3.1	Introduction	21
3.2	Preliminaries	23
3.2.1	Definitions and properties on graphs	23
3.2.2	Vertex set properties and constraints on graphs	26
3.2.3	Paths in a directed acyclic graph	27
3.3	Local properties and enumeration for linearly ordered graphs	30
3.4	MDS, MCDS and MIR in interval graphs	33
3.4.1	Interval graphs and locally definable basic properties	33
3.4.2	Interval graphs and minimal dominating sets	35
3.4.3	Interval graphs and minimal connected dominating sets	37
3.4.4	Interval graphs and maximal irredundant sets	39
3.5	MDS, MCDS and MIR in permutation graphs	41

3.5.1	Permutation graphs and locally definable basic properties	41
3.5.2	Permutation graphs and minimal dominating sets	47
3.5.3	Permutation graphs and minimal connected dominating sets	48
3.5.4	Permutation graphs and maximal irredundant sets	50
3.6	Local properties and enumeration for cyclically ordered graphs	53
3.7	MDS and MIR in circular-arc and circular-permutation graphs	57
3.7.1	Enumeration in circular-arc graphs	57
3.7.2	Enumeration in circular-permutation graphs	60
3.8	Conclusion	61

<p>4 Efficient enumeration of k-arc-connected orientations in graphs</p>
--

4.1	Introduction	63
4.2	Preliminaries	67
4.2.1	Notations and basics of orientations	68
4.2.2	Frank’s theorem on k -arc-connected orientations	69
4.3	Enumeration of k -connected orientations via submodular flows	74
4.3.1	Partial orientation extension with submodular flow feasibility	75
4.3.2	Partial orientation extension using minimum-cost orientation	77
4.4	Orientations with prescribed outdegree sequence	78
4.4.1	Flashlight backtrack search via integer points of polytopes	78
4.4.2	Flashlight backtrack search via cycle flips	88
4.5	k -connected outdegree sequences via path flips	90
4.5.1	Reverse search	93
4.5.2	Flashlight backtrack search via path flips	96
4.6	Simple and modular enumeration of k -connected orientations	99
4.7	Conclusion	102

<p>5 Conclusion</p>
--

Introduction

The thesis focuses on efficient algorithms for some enumeration problems in graphs. The stakes are double and of independent interest: enumerating *locally definable vertex set properties* and enumerating *k-arc-connected orientations*. We will analyse the efficiency of the algorithms in an *output-sensitive way* and pay particular attention to the *delay* between two consecutive solutions.

In this introductory chapter we first give an overview of measures of complexity and algorithmic methods associated to enumeration theory. Then, we present the contributions and finally we detail the structure of the thesis.

1.1 Enumeration: algorithms and complexity

Algorithmics is one of the domains whose importance has considerably increased during the last decades. The main problem of this area is to find efficient algorithms to solve a given problem. Measuring efficiency then depends on the computational task we address. We can distinguish several types of problems, in which the computational goals differ.

- *Decision problems*: decide whether an instance of the problem has a solution or not;
- *Search problems*: find a solution (if there is one) and return it;
- *Counting problems*: count how many solutions an instance of the problem has;
- *Enumeration problems*: produce all the solutions of an instance.

In many domains, enumeration problems are the most natural kind of problems: generating objects with specified properties has important applications in computer science (data mining, machine learning, artificial intelligence) as well as in other sciences, especially biology. Since the number of solutions to a problem might be exponential, at first, computer limitations put the field of enumeration theory aside. The question of enumerating all solutions of a given search problem arose only for relatively simple combinatorial objects. Gradually, with the technological development, the nature of the considered objects, became more and more complex. Enumeration experienced renewed interest and became a full domain of algorithmics and complexity theory.

Usual measures of complexity are rather inadequate when one focuses on enumeration. One reason is the potentially exponential number of objects to be enumerated. It renders inappropriate the traditional input-sensitive approach, which estimates efficiency by relating the total time needed to generate all the solutions to the size of the input. In contrast, the output-sensitive approach measures the time complexity of an enumeration algorithm taking into account both the size of the input and that of the output. An

enumeration algorithm with total running time bounded by a polynomial depending both on the size of the input and the size of the output is called an *output-polynomial* algorithm. The *delay* between the successive outputs is a finer and more relevant measure of efficiency for enumeration, which measures the regularity of the production of solutions. It is natural to aim at controlling this maximum delay or at least the average over all delays, called the amortized time. As for decision problems, an acceptable delay is one that is polynomially bounded in the size of the input. We can hope to obtain finer results (linear or constant delay), for example by using some pre-processing steps. In database theory the query answering problem, which aims at producing all answers to a query without duplicates, is an example of enumeration problem that can be solved with constant delay after a linear pre-processing [29]. Another constant delay algorithm is the Gray code for enumerating all bit vectors of given length. An amortized analysis guarantees the average performance of each operation in the worst case. Amortized analysis is used to analyse certain algorithms that perform a sequence of similar operations. A worst-case analysis, in which we sum the worst-case times of the individual operations, may be unduly pessimistic, because it ignores correlated effects of the operations on the data structure. Instead of bounding the cost of the sequence of operations by bounding the actual cost of each operation separately, an amortized analysis provides a bound on the actual cost of the entire sequence. One advantage of this approach is that although some operations might be expensive, many others might be cheap. In other words, many of the operations might execute much faster than the worst-case estimation time.

Many of the strategies used for enumeration rely on the solution space. The idea is to highlight a property or structure of that set of solutions, and then leveraging it to enumerate the solutions more efficiently. In many cases, the solution space is structured using a graph whose vertices are solutions. Enumeration then boils down to an intelligent search of the graph where the challenge is to avoid two issues: explore areas of space in which there is no solution and output solutions already considered. To meet these constraints, there are two different types of structures that can be exploited, corresponding respectively to the so-called *Backtrack Search* and *Reverse Search* methods.

A backtrack search algorithm relies on a tree in which only leaves represent the solutions. Each node corresponds to an embryo of solution that is gradually extended to get complete solutions. The calculation is then like traversing a tree built from the root to the leaves. Each visited leaf potentially corresponds to a solution. Rather than starting from scratch for calculating each new solution, backtracking to explore a new branch saves time. This corresponds to a depth-first search. The interest of this method is that, by construction, it avoids repetitions. A backtrack search is for example successfully used, after a linear pre-processing, for the generation of paths of a directed acyclic graph [66]. On the other hand, for certain problems, there is the risk of entering branches that contain no solution. In this case, one descends deeper and deeper while the target leaf is unsuitable. Hence, we need an intelligent search of the solution space. This can be done provided that we dispose of an oracle that decide if a branch is worth exploring or not. In other words, we must be able to check before each junction that the track that we are about to follow will be fruitful.

In the reverse search method, introduced by Avis and Fukuda [3], one exploits a graph whose nodes (and not only leaves) represent the solutions. This graph is built on the fly in following a strategy based on some proximity of the solutions to a predefined goal. In other words, each solution is connected to others considered as realizing more effectively a certain objective. This neighbourhood link provides the whole solutions with an oriented graph structure that we can go through to enumerate the solutions. To build this graph on the fly we need to define an adjacency oracle which gives the neighbours of a vertex, as well as a local search procedure. This local search procedure is used to traverse the graph efficiently and provides a spanning tree. The delay of the enumeration algorithm obtained by this reverse search technique is bounded according to the height of this spanning tree. It turns out that for many problems, this height is polynomial in the size of the input, and thus the corresponding algorithm has then polynomial delay.

It is for example the case for the enumeration of connected induced subgraphs or for the enumeration of topological orders of a graph [3]. Another example of a reverse search algorithm is the one proposed by Makino and Uno for enumerating maximal cliques and maximal bipartite cliques [77].

Another technique for finding efficient enumeration algorithms is to use an appropriate notion of reduction. This is a very classical method for decision problems. The idea is to reduce a given problem to a well-known problem for which efficient algorithms are known. Roughly speaking a reduction is a way of turning one problem into another so that the second problem can be used to solve the first one. Karp reductions as well as Turing reductions serve this purpose for decision problems. For example, in [34] it is shown that the problem of finding an orientation of a graph with prescribed outdegree at each vertex reduces to a flow-problem, which is known to be solvable in polynomial time (see for example [84]). For counting problems, reductions from one problem to the other have to preserve not only the existence of a solution but also the number of solutions. In [97] Valiant defined such a reduction, referred to as *parsimonious reduction*. An appropriate reduction between enumeration problems has to carry even more information since it has to preserve the sets of all solutions (and not only their cardinality). In the next section we will define a *polynomial enumeration reduction* that is analogous to the one defined by Bagan *et al.* [4]. It establishes a one-to-one correspondence between sets of solutions and thus is very strict. It is stricter than other reductions that appeared in the literature (see e.g. [79] and [26]). Nevertheless, since in this thesis we will be interested not only in polynomial delay but also in linear delay, we will actually need an even tighter notion of reduction, which preserves the existence of a linear delay algorithm.

In this work, we will illustrate each of these different enumeration techniques - backtrack search, reverse search and reduction - through problems related to graphs.

1.2 Contributions of the thesis

The present work addresses two independent issues, both related to enumeration in graph theory: the enumeration of vertex set properties and the enumeration of k -arc-connected orientations. In the following, given a graph $G = (V, E)$, we will always have $|V| = n$ and $|E| = m$.

Enumerating sets of different kinds in graphs ranges among the classical problems of combinatorics. There is a variety of measures to describe the structure of graphs, such as the diameter, radius, minimal and maximal degrees, the eigenvalues and planarity. Besides their obvious theoretical value, enumeration problems in connection with graphs have proved to be of a certain interest in many areas. For instance, graphs provide a simple and understandable yet powerful tool to describe the structure of molecules. The chemist Hosoya [61] investigated the surprising relation between the physicochemical properties of a molecule and the number of its independent edge subsets (matchings). Biologists use models like graphs as abstractions of reality. The nodes typically describe proteins, metabolites, genes, or other biological entities, whereas the edges represent various types of interactions between the nodes. For instance, protein-protein interaction networks are conveniently modelled by undirected graphs, where the nodes are proteins and two nodes are connected by an undirected edge if the corresponding proteins physically bind [2, 69]. Enumeration algorithms are particularly useful whenever the goal of a problem is not clear and all its solutions need to be checked. Since one peculiar property of biological networks is the uncertainty, a scenario in which enumeration algorithms can be helpful is biological network analysis [78].

A famous example of enumeration problem in graph theory is that of enumerating the minimal dominating sets of a graph. This problem is particularly interesting since it has been proved by Kanté *et al.* [65] that the longstanding open question whether there is an output-polynomial algorithm to enumerate minimal transversals of a hypergraph is equivalent to the question whether there exists an output-polynomial algorithm to enumerate minimal dominating sets of a graph. Although the question remains open in general, a large number of positive results has been obtained on particular graph classes (see e.g. [11, 23, 24, 38, 52]).

In particular it has been shown in [66] that one can enumerate minimal dominating sets with linear delay on interval and permutation graphs. One of the main objectives of the thesis is to build on this result in order to extend it to a much broader context. In Chapter 3 we propose a formal framework that allows us to deal not only with one property and a graph class, but with a whole family of properties and a whole family of graphs. This broader vision enables us to identify structures that allows for linear delay enumeration for a very large class of graphs and problems. In using a reduction to the enumeration of paths in a directed acyclic graph, we put forward a general method for finding linear delay algorithms for the enumeration of vertex set properties in graphs. To do so, we consider graphs in which there is a natural order on the vertices and we say that a property is locally definable with respect to this order if one can decide if a subset of vertices respects the property by scanning its ordered list of vertices through a sliding window of fixed size and verifying that each scanned tuple satisfies some constraints. Our main result is that such a locally definable vertex set property can be enumerated with linear delay after a polynomial pre-processing.

Another example of interest in enumeration graph theory is the generation of the orientations with given properties. Given an undirected graph G , an orientation is a directed graph (sometimes also called digraph) obtained by assigning directions to the edges of G . Orientations have their important part to play in network theory, which is largely inspired by real-world networks such as social, biological and technological networks where patterns of connections between their elements are displayed neither purely regular nor purely random.

The notion of strong connectivity is very natural since it means that every node can be reached from every node. Such a constraint explains the natural applications of strong orientations to transportation networks, which describe a flow structure of some commodity, like vehicular movements, pipelines, aqueducts or power lines. The necessary and sufficient condition of the existence of a strong orientation was given by Robbins and illustrated through the modeling of the traffic system of a town [87]. Vertices of the graph stand for street intersections and two vertices are joined by an edge if it is possible to travel from one intersection to the other without crossing a third intersection. During the week, all streets are two-ways, but we need to be able to repair any given street at a time and still preserve traffic allowing people to reach any part of the town from any other part. On week-ends, all streets are available because no repairing is done but the streets are only one-way and again we still want to be able to travel from any point of the city to any other. Robbin's theorem states that such a system of roads is suitable during the week if and only if it is suitable during the week-end. In other words, there exists a strong orientation of an undirected graph if and only if the graph is 2-edge connected, i.e., any removal of a single edge does not disconnect the graph. Robbin's theorem has been extended by Boesch and Tindell to mixed graph [10], and in [18] Chung et al. describe a linear time algorithm for finding a strong orientation in a mixed multigraph. Moreover, the enumeration of all strong orientations is doable with linear delay [21].

In this thesis, we will be interested in a generalization of strong orientations, namely k -arc-connected orientations. The main objective of Chapter 4 is to give a simple algorithm for the enumeration of the k -arc-connected orientations of G , i.e., those where at least k arcs have to be removed in order to destroy the strong connectivity. The concept of k -arc-connectivity is classic in the theory of directed graphs. The problem of producing a k -arc-connected orientation, if there exists one, has been studied in [41, 43, 62] and [74]. However, to our knowledge the enumeration of k -arc-connected orientations has not been studied explicitly for $k \geq 2$. We will solve this problem by relying on results by Frank, who made several important contributions to the theory of k -arc-connected orientations (see for example [39–42]). In particular he showed that any two k -arc-connected orientations of some graph can be transformed into each other by reversing directed paths and directed cycles [40].

We will split the problem of enumerating all k -arc-connected orientations of a graph into two subproblems of independent interest. The first one is the enumeration of all orientations respecting some fixed

outdegree sequence and the second one is the enumeration of all possible outdegree sequences that can be attained by a k -arc-connected orientation. For each of these subproblems, we will propose and compare different solving methods.

1.3 Structure and content of the thesis

The thesis is structured as follows. Chapter 2 sets the context of enumeration complexity. The main complexity classes are introduced in Section 2.1. Section 2.2 is dedicated to the algorithmic techniques associated to enumeration: flashlight backtrack search, reverse search and reduction are defined. Each of these techniques will be illustrated in the sections that follow. Finally, a presentation of hardness results is given in Section 2.3.

In Chapter 3 we present the first main result of the thesis, namely that locally definable vertex set properties can be enumerated with linear delay after a polynomial pre-processing. This result is obtained through a reduction to the problem of enumerating the paths of a directed acyclic graph and is illustrated through two examples: the enumeration of the minimal (connected) dominating sets and the enumeration of the maximal irredundant sets of a given graph. Thus, after an introduction of the problem in Section 3.1, Section 3.2 is dedicated to the study of the relation between minimal dominating sets and maximal irredundant sets. More precisely, we propose a characterization of the graphs that respect for each of their induced subgraph the property that the set of minimal dominating sets and the set of maximal irredundant sets correspond. Furthermore, in this section we also recall the folklore result that all paths of a directed acyclic graph can be generated in linear time. Locally definable vertex set properties are treated in the case of linearly ordered graphs in Section 3.3, then Sections 3.4 and 3.5 are dedicated to applications in interval and permutation graphs respectively. The case of locally definable vertex set properties in cyclically ordered graphs is treated in Section 3.6. Section 3.7 is devoted to applications in circular-arc and circular-permutation graphs. We conclude this chapter in Section 3.8.

Chapter 4 is dedicated to the second main result of the thesis, namely the enumeration of k -arc-connected orientations. The context is presented in Section 4.1. The different algorithms presented to solve the k -arc-connected orientations problem are all based on a result by Frank. Thus, we recall and analyse the latter in Section 4.2. Then, in Section 4.3, a first polynomial delay algorithm for generating all k -arc-connected orientations is presented; it is a reduction to the submodular flow problem. The complete understanding of the proofs behind this algorithm being rather intricate and the implementation not elementary, we propose in the two following sections an alternative method to solve the problem. The enumeration of the α -orientations for a fixed α is treated in Section 4.4 and the enumeration of the outdegree sequences for which there exists a k -arc-connected orientation is treated in Section 4.5. We finally combine the two algorithms in Section 4.6, thus obtaining an algorithm with a better delay and a much better amortized time than the one through reduction to the submodular flow problem. We conclude this chapter in Section 4.7.

In Chapter 5 we conclude the thesis with some general final remarks.

2

Complexity for enumeration

Contents

2.1	Complexity classes for enumeration	7
2.2	Methods for efficient enumeration	11
2.2.1	Flashlight Backtrack Search	11
2.2.2	Reverse Search	12
2.2.3	Reductions	15
2.3	Hardness results	16

This chapter contains some preliminaries concerning the complexity of enumeration. In the three sections to come we present successively the main complexity classes, algorithmic techniques and hardness results associated to enumeration.

2.1 Complexity classes for enumeration

Let Σ be a finite alphabet and R a polynomially bounded binary relation on Σ^* , i.e., there exists a polynomial p such that for all $(x, y) \in R$, $|y| = O(p(|x|))$. To such a relation one can associate an enumeration problem:

ENUM(R) <i>Input</i> : $x \in \Sigma^*$ <i>Output</i> : $R(x) = \{y \in \Sigma^* \mid (x, y) \in R\}$

Given $x \in \Sigma^*$ four computational tasks are naturally involved in the enumeration of $R(x)$: deciding whether $R(x) \neq \emptyset$; deciding whether some partial answer can be extended to a solution; deciding whether a current set of solutions can be completed and eventually finding another solution.

DECIDE(R) <i>Input</i> : $x \in \Sigma^*$ <i>Question</i> : is there $y \in \Sigma^*$ such that $y \in R(x)$?
--

EXTSOL(R)

Input : $(x, y) \in \Sigma^* \times \Sigma^*$

Question : is there $y' \in \Sigma^*$ such that $yy' \in R(x)$?

ALLSOL(R)

Input : $x \in \Sigma^*$ and $S \subseteq R(x)$

Question : $S = R(x)$?

ANOTHERSOL(R)

Input : $x \in \Sigma^*$ and $S \subseteq R(x)$

Output : $y \in R(x) \setminus S$ if such a y exists, NIL otherwise.

Enumeration algorithms output all solutions without repetition. As the number of solutions may be exponential, a polynomial bound on the time needed for the whole computation is not realistic. Tractability is rather captured by polynomial bounds in the size of both the input and the output. Such an approach is said to be *output-sensitive*. The *delay* of an enumeration algorithm is the maximal time elapsed between the production of two consecutive solutions (including the time to find the first solution and the time to detect that no further solution exists).

Definition 1. Let y_1, \dots, y_n be the solutions of some enumeration problem on the instance x enumerated in this order by an algorithm \mathcal{A} . For $1 \leq i \leq n-1$, let $T(\mathcal{A}, i)$ be the time required by \mathcal{A} until it outputs y_i , and $T(\mathcal{A}, n)$ be the time required by \mathcal{A} until it outputs y_n and stops. Then $\text{Delay}(\mathcal{A}, 1) = T(\mathcal{A}, 1)$ (referred to as the pre-processing time) and for $2 \leq i \leq n$, $\text{Delay}(\mathcal{A}, i) = T(\mathcal{A}, i) - T(\mathcal{A}, i-1)$ (referred to as the i -th delay).

The three central complexity classes in the classification of enumeration problems are TotalP, IncP and DelayP. The first one, *Total Polynomial*, is the class of problems for which there exist algorithms whose total running time is bounded by a polynomial in the size of the input and the number of solutions. Such algorithms can be efficient for problems with a small number of solutions, but their running time may become unreasonable when the number of solutions approaches exponential. With complexity class TotalP, one can expect the first solution for a very long time, which can be problematic. More generally regularity of the production of the solutions is an important feature. Indeed, when solutions are produced regularly then they can be treated as you go, moreover within a fixed amount of time we are sure to obtain a certain number of solutions. For this reason, refinements of TotalP have been introduced, these are complexity classes IncP and DelayP. The class IncP, or *Incrementally Polynomial*, combines those problems that are solvable with a polynomial incremental delay, that is to say polynomial in the size of the input and the number of solutions already generated. The even stronger complexity class DelayP, or *Polynomial Time Delay*, stands for enumeration problems which are solvable with a delay polynomial in the size of the input. In this thesis, we will also be interested in an average over all delays, called *amortized time*. Instead of bounding the cost of a sequence of delays by bounding the cost of each delay separately, we want to guarantee the average performance of each delay in the worst case. The class AmortizedP groups such problems for which the average over all delays is bounded by a polynomial in the size of the input.

These classes are formally defined as follows and the hierarchy is illustrated in Figure 2.1.

Definition 2. Let $\text{ENUM}(\mathbb{R})$ be an enumeration problem and for an instance x , let n denote the number of solutions, $n = |\mathbb{R}(x)|$. The enumeration problem $\text{ENUM}(\mathbb{R})$ belongs to:

- TotalP if there exists an enumeration algorithm \mathcal{A} for $\text{ENUM}(\mathbb{R})$ and a polynomial p such that on each instance x , the total time of \mathcal{A} is bounded by $p(|x|, n)$.
- IncP if there exists an enumeration algorithm \mathcal{A} for $\text{ENUM}(\mathbb{R})$ and a polynomial p such that on each instance x , for $i \in \{1, \dots, n\}$, $\text{Delay}(\mathcal{A}, i)$ is bounded by $p(|x|, i)$.
- DelayP if there exists an enumeration algorithm \mathcal{A} for $\text{ENUM}(\mathbb{R})$ and a polynomial p such that on each instance x , for $i \in \{1, \dots, n\}$, $\text{Delay}(\mathcal{A}, i)$ is bounded by $p(|x|)$.
- AmortizedP if there exists an enumeration algorithm \mathcal{A} for $\text{ENUM}(\mathbb{R})$ and a polynomial p such that on each instance x , the total time of \mathcal{A} is bounded by $n \cdot p(|x|)$.

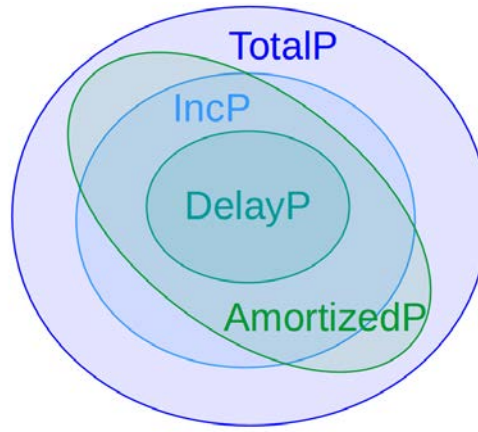


Figure 2.1: Enumeration Complexity Hierarchy

The problem ANOTHERSOL is closely related to the class IncP, it bridges the complexity of a search problem with the complexity of enumeration. An example of enumeration problem that belongs to complexity class IncP and that leans on its associated ANOTHERSOL problem comes from databases. Given a relational schema, the goal is to enumerate all the minimum keys. The belonging of this problem to the class IncP is implicitly proved by Lucchesi and Osborn in [76]. Their analysis is done in terms of total execution time, without paying attention to the delay which turns out to be incremental.

As Mary and Strozecki make explicit in [80], many problems which can be solved with an incremental delay have the following form: given a set of elements and a polynomial time function acting on tuples of elements, produce the closure of the set by the function. For instance, the best algorithm to generate all circuits of a matroid is in incremental delay because it uses some closure property of the circuits, see the paper by Khachiyan *et al.* [68].

Polynomial delay algorithms have attracted a lot of attention. In the age of the internet and huge data transfers a small delay between successive solutions ensures a regularity in the enumeration process and a continuous data flow. The decision problem EXTSOL is closely related to the class DelayP since the existence of a polynomial algorithm for EXTSOL(\mathbb{R}) implies the existence of a polynomial delay algorithm for $\text{ENUM}(\mathbb{R})$, see [80] and Subsection 2.2.1.

In [80], Mary and Strozecki address the problem of generating all elements obtained by the saturation of an initial set by some operations and they try to understand when saturation problems, which are natural incremental delay problems, can be solved by a polynomial delay algorithm. They present an algorithm that computes with polynomial delay the closure of a family of sets by any set of "set operations" (e.g. by union, intersection, difference, symmetric difference...). To do so, they prove that for any set of operations \mathcal{F} , it is possible to decide in polynomial time whether an element belongs to the closure by \mathcal{F} of a family of sets. Moreover, they prove that when the relation is over a domain larger than two elements, the generic enumeration method fails, since the associated decision problem is NP-hard.

About enumeration problems that belong to complexity class DelayP, we know for example from Johnson *et al.* in [63] that the maximal independent sets of a graph can be generated with polynomial delay. In [64], Kanté *et al.* propose a polynomial delay algorithm with polynomial space for the generation of minimal dominating sets in split graphs. In [25] Creignou and Hébrard show a dichotomy classification for the enumeration problem associated with generalized satisfiability. They prove that there exists a class of generalized satisfiability problems \mathcal{G} such that every problem in \mathcal{G} belongs to DelayP whereas for any generalized satisfiability problem not belonging to \mathcal{G} a polynomial delay algorithm which generates all the solutions does not exist unless $P = NP$. Class \mathcal{G} consists of problems equivalent to the problem of satisfiability for a conjunction of Horn clauses, anti-Horn clauses, 2-clauses or XOR-clauses.

Class AmortizedP groups problems for which the average over all delays is small, even though a single delay between two consecutive solutions might be expensive. The amortized cost cannot be higher than the maximum delay, but may very well be lower: an example is the algorithm proposed in [21] to enumerate all strong orientations of a given graph. Here the delay is $O(m^2)$ and the amortized time is $O(m)$. Another example will be our algorithm for generating all k -arc-connected orientations of a graph. The delay is $O(knm^2)$ while the amortized time is $O(m^2)$. Amortized analysis is not just an analysis tool, it is also a way of thinking about the design of algorithms, since the design of an algorithm and the analysis of its running time are often closely intertwined. Thus, when we perform an amortized analysis, we often gain insight into a particular data structure, and this insight can help us optimize the design.

In this thesis we will be interested in problems in DelayP. For such problems we will sometimes propose algorithms with *linear delay*.

Definition 3. *With the notations of Definition 1, an enumeration algorithm \mathcal{A} has linear delay if $\text{Delay}(\mathcal{A}, 1)$ is bounded by a polynomial in the size of the input, and for $2 \leq i \leq n$, $\text{Delay}(\mathcal{A}, i)$ is bounded by a linear function in $(|y_{i-1}| + |y_i|)$.*

In some sense linear delay is the best we can hope whenever we deal with enumeration problems in which we want to print all solutions. In fact, an algorithm which enumerates a set \mathcal{C} in linear delay outputs \mathcal{C} in total time $O(p(|x|) + \sum_{C \in \mathcal{C}} |C|)$ where $p(|x|)$ is the polynomial bounding the pre-processing time. Roughly speaking such an algorithm can be considered as optimal since after a pre-processing step the time required is not more (up to a multiplicative constant) than the time needed to write up the solutions.

Such linear delay enumeration algorithms appear in particular in the enumeration of vertex set properties in graphs. We already mentioned the result by Kanté *et al.* in [66] where an algorithm is stated for the enumeration of minimal dominating sets in interval and permutation graphs. In [99] Kanté *et al.* improve this result by showing that all minimal dominating sets of graphs of bounded linear maximum induced matching width can be enumerated with linear delay using polynomial space.

In Chapter 3, we will pay special attention to this notion of linear delay.

2.2 Methods for efficient enumeration

We present here a non-exhaustive list of algorithmic techniques associated to enumeration. Flashlight backtrack search, reverse search and the notion of reduction in the framework of enumeration will be explained. Each of these techniques will be illustrated in the chapters to come.

2.2.1 Flashlight Backtrack Search

The *Backtrack Search* method consists in going through a tree whose vertices are partial candidate solutions and some leaves are solutions. The partial candidate solutions are gradually extended to get complete solutions. Going through such a graph by a depth-first search allows the enumeration of all the solutions. In full generality, the delay of this algorithm can be exponential. In order to obtain a polynomial delay, it is important that no branch is searched in vain, that is, any explored internal node must be the prefix of at least one final solution. This method is then called *Flashlight Backtrack Search*; it is based on EXTSOL and we have the following well-known result.

Proposition 4 ([80]). *Let Σ be a finite alphabet and R be a polynomially bounded binary relation on Σ^* . If $\text{EXTSOL}(R) \in \text{P}$, then $\text{ENUM}(R) \in \text{DelayP}$.*

The proof of this proposition boils down to the following algorithm, in which ε denotes the empty word. Without loss of generality, we will consider that all solutions of $R(x)$ have the same size $p(|x|)$.

Algorithm 1: Flashlight backtrack search method
<p>Input: $x \in \Sigma^*$ with $p(x)$ the size of a solution Output: $R(x)$</p> <pre style="margin: 0;"> 1 if DECIDE(R)(x) then 2 Generate ($x, \varepsilon, p(x)$) 3 Function Generate ($x \in \Sigma^*, y \in \Sigma^*, l \in \mathbb{N}$): 4 if $y < l$ then 5 for each $a \in \Sigma$ do 6 if EXTSOL(x, ya) then 7 Generate (x, ya, l) 8 else 9 Output y </pre>

This algorithm goes through a tree of depth $p(|x|)$ and of degree $|\Sigma|$ and enumerates the leaves without repetition. In fact, in the tree built by a flashlight backtrack search, a node generates branches with disjoint partial solutions. The algorithm does not go back on already treated partial solutions, i.e., each of these partial solutions will not be modified in the rest of the tree, it will only grow until a final solution is found. Thanks to the call to EXTSOL, no branch is explored in vain. Thus, the delay of this flashlight backtrack search algorithm for $\text{ENUM}(R)$ is $O(p(|x|) \cdot f(|x|))$ where f denotes the complexity of $\text{EXTSOL}(R)$.

We already mentioned that the models of a Horn formula can be generated with polynomial delay [25]. This bound can be obtained thanks to the flashlight backtrack search method. Since the satisfiability of a Horn formula is decidable in polynomial time and since a Horn formula in which some variables are interpreted to true or false is still a Horn formula, a natural way to generate all models of a Horn formula is to try possible valuations of a given variable and check whether the obtained formula is satisfiable. If it is, then we recurse.

2.2.2 Reverse Search

The reverse search method has been proposed by Avis and Fukuda in [3]. It is a general recipe to construct tree-structured enumeration methods useful for enumerating combinatorial sets. This powerful technique relies on the structure of the solution space and not on the tractability of the extension problem $\text{EXTSOL}(\mathcal{R})$. The idea is to go through a graph built on the fly whose vertices are the objects to be enumerated. This graph is often referred to as a metagraph. To describe this metagraph, we have an *adjacency oracle* which gives the neighbours of a vertex, as well as a *local search procedure*, which allows the efficient traversal of the graph.

As the solution graph is not explicitly given, the adjacency oracle helps with its exploration: it allows us to list the neighbours of a given vertex.

Definition 5. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a connected graph whose vertices are the objects to generate and $\bar{\Delta}$ a positive integer. The adjacency oracle Adj fulfills the following requirements:

- For each $v \in \mathcal{V}$ and each $p \in \{1, \dots, \bar{\Delta}\}$, the oracle returns $Adj(v, p)$, which is either a neighbour of v or 0.
- If $Adj(v, p) = Adj(v, p') \neq 0$, then $p = p'$.
- For all $v \in \mathcal{V}$, $\{Adj(v, p) \mid Adj(v, p) \neq 0 \text{ and } 1 \leq p \leq \bar{\Delta}\}$ is exactly the set of vertices adjacent to v .

These three conditions imply that Adj returns each adjacent vertex of v exactly once during the $\bar{\Delta}$ inquiries $Adj(v, p)$, $1 \leq p \leq \bar{\Delta}$, for each vertex v .

The local search function is an algorithm for finding one solution. Let $S \subseteq \mathcal{V}$ be called a *sink*. A *local search* on \mathcal{G} is a procedure that allows to reach from any vertex of the metagraph \mathcal{G} , a vertex of the sink S .

Definition 6. A triplet (\mathcal{G}, S, f) is a finite local search if $S \subseteq \mathcal{V}$ and $f : \mathcal{V} \setminus S \rightarrow \mathcal{V}$ satisfies the following statements:

- For all $v \in \mathcal{V} \setminus S$, $(v, f(v)) \in \mathcal{E}$;
- For all $v \in \mathcal{V} \setminus S$, there exists an integer l such that $f^l(v) \in S$.

The idea behind the algorithm Local Search is that as long as a vertex does not belong to S , we apply the function f :

Algorithm 2: Local search algorithm
<p>Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, $S \subseteq \mathcal{V}$, $f : \mathcal{V} \setminus S \rightarrow \mathcal{V}$, $v_0 \in \mathcal{V}$</p> <pre style="margin: 0;"> 1 $v \leftarrow v_0$; 2 while $v \notin S$ do 3 $v \leftarrow f(v)$; 4 Output v</pre>

We have seen that a local search traverses the graph to the sink. We call the *trace* of a local search (\mathcal{G}, S, f) a directed subgraph $T = (\mathcal{V}, \mathcal{E}(f))$ of \mathcal{G} , with $\mathcal{E}(f) = \{(v, f(v)) \mid v \in \mathcal{V} \setminus S\}$. The trace T is simply the directed graph with all vertices of \mathcal{G} and those edges of \mathcal{G} used by the local search.

Proposition 7. *If (\mathcal{G}, S, f) is a local search, then the trace T is a spanning forest of \mathcal{G} with each tree rooted in a vertex of S .*

Once the adjacency oracle and local search are defined, the graph \mathcal{G} can be traversed by applying the algorithm *Reverse Search* described below. Unlike the local search that traverses the graph *up to* a sink, reverse search traverses the graph starting *from* a sink.

Algorithm 3: Reverse search algorithm

```

Input:  $Adj, \bar{\Delta}, S, f$ 
1 for each  $s \in S$  do
2    $v \leftarrow s$ 
3    $j \leftarrow 0$ 
4   repeat
5     while  $j < \bar{\Delta}$  do
6        $j \leftarrow j + 1$ 
7        $next \leftarrow Adj(v, j)$ 
8       if  $next \neq 0$  then
9         if  $f(next) = v$  then
10           $v \leftarrow next$ 
11           $j \leftarrow 0$ 
12     if  $v \neq s$  then
13        $u \leftarrow v$ 
14       Output  $v$ 
15        $v \leftarrow f(v)$ 
16        $j \leftarrow 0$ 
17       repeat
18          $j \leftarrow j + 1$ 
19       until  $Adj(v, j) = u$ ;
20   until  $v = s$  and  $j = \bar{\Delta}$ ;

```

The *repeat* at line 4 is divided into two parts: a loop *while* and a condition *if*. The loop *while*, for j and v fixed, points to the first neighbour of v whose index corresponds to j with respect to the neighbourhood function f . This operation is then repeated on this neighbour of v and so on until the end of the first branch of the first spanning tree is reached. The vertex corresponding to this end is then enumerated. The condition *if* at line 12 locates the index of the last vertex and then goes back in the branch by appealing again to the loop *while* in order to find the next branch of the spanning tree to generate its extremity. It is a post-fixed enumeration, see Figure 2.2 for an illustration.

Let us now analyse the complexity of Algorithm 3 above. We denote by t_{Adj} and t_f respectively the calculation time of $Adj(v, j)$ for all v, j and of $f(v)$ for all v . The complexity of the loop *while* (line 5) is in $O(\bar{\Delta} \cdot (t_{Adj} + t_f))$. The complexity of the condition *if* (line 12-19) is in $O(t_f + \bar{\Delta} \cdot t_{Adj})$. These two steps are repeated in the worst case $|\mathcal{V}|$ times. The total running time of *Reverse Search* is therefore:

$$O(|\mathcal{V}| \cdot \bar{\Delta} \cdot (t_{Adj} + t_f))$$

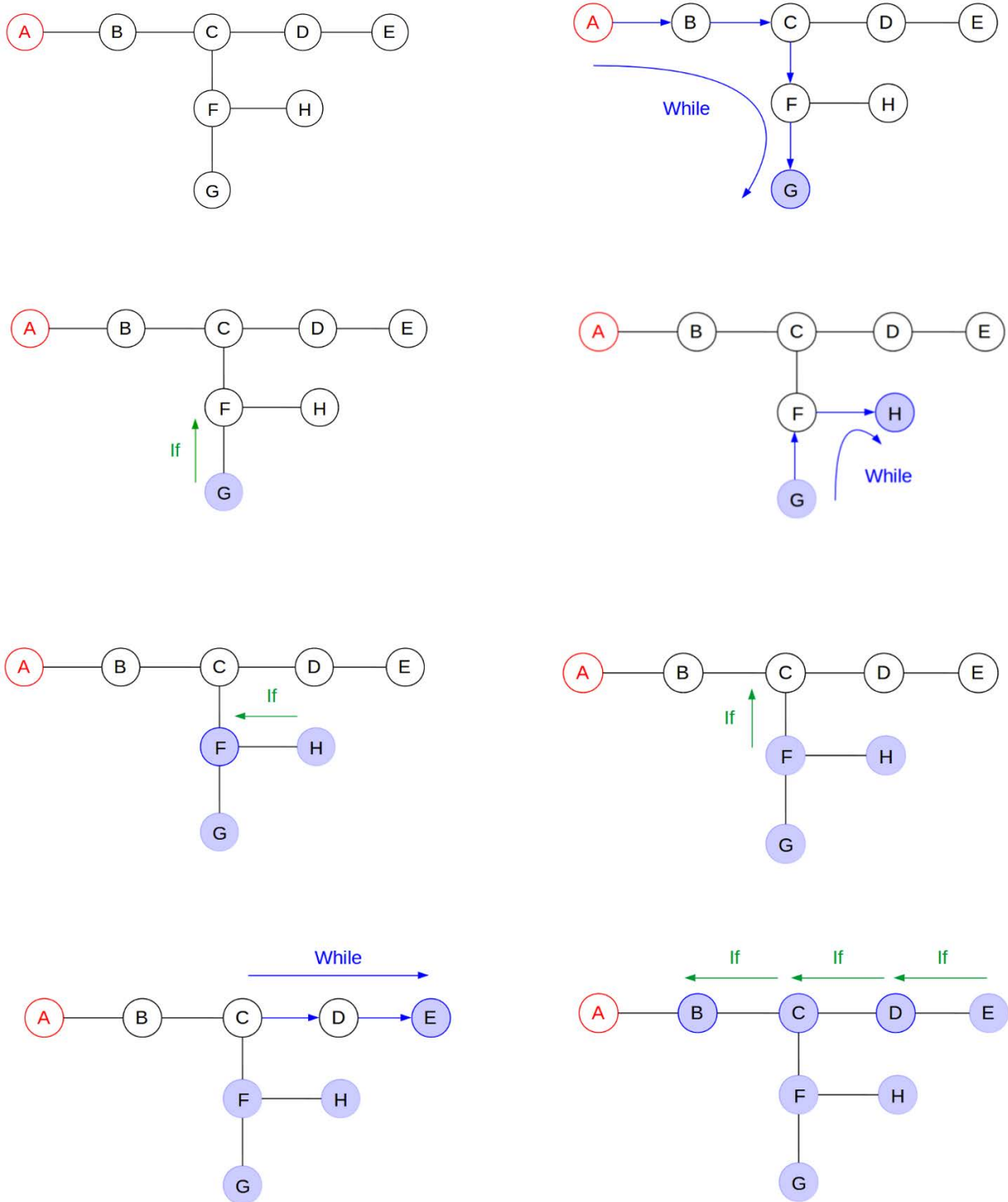


Figure 2.2: Reverse Search with in red the sink and in blue the output vertices

The delay of the algorithm is bounded not according to the total number of solutions but according to h the height of the spanning tree of \mathcal{G} described by f :

$$O(h \cdot \bar{\Delta} \cdot (t_{Adj} + t_f))$$

Indeed, a solution is enumerated once the first branch of the first spanning tree is traversed. This corresponds to the course of the *while* loop, see Figure 2.2. It turns out that for many problems, h is polynomial in the size of the input, and the corresponding algorithm is then polynomial delay.

Reverse search is not quite an algorithm, rather it is a general construction principle that is applicable to a wide variety of problems and often leads to optimal algorithms for enumeration problems. Broad applications of reverse search are developed by Avis and Fukuda in [3] to various problems in operations research, combinatorics and geometry. In particular, algorithms are presented for the generation of connected induced subgraphs or spanning trees of a given graph, for the listing of triangulations of a set of n points in the plane, for the listing of cells in a hyperplane arrangement in \mathbb{R}^d or for the enumeration of topological orderings of an acyclic graph.

2.2.3 Reductions

Roughly speaking a reduction between two enumeration problems $\text{ENUM}(R_1)$ and $\text{ENUM}(R_2)$ allows us to solve $\text{ENUM}(R_1)$ in using an algorithm that solves $\text{ENUM}(R_2)$. Having such a reduction at hand, reducing some enumeration problem to a well-studied enumeration problem for which efficient enumeration algorithms exist, is a classical algorithmic method. There are several notions of reductions for enumeration that appear in the literature. Here we define a polynomial enumeration reduction that is analogous to the one defined by Bagan *et al.* in [4].

Definition 8 (polynomial enumeration reduction). *Let $\text{ENUM}(R_1)$ and $\text{ENUM}(R_2)$ be two enumeration problems. A polynomial enumeration reduction from $\text{ENUM}(R_1)$ to $\text{ENUM}(R_2)$ is a pair of mappings (σ, τ) with $\sigma : \Sigma^* \rightarrow \Sigma^*$ and $\tau : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ which satisfies:*

- σ and τ are computable in polynomial time;
- for every $x \in \Sigma^*$, the mapping $y \mapsto \tau(\sigma(x), y)$ is one-to-one mapping from the set $R_2(\sigma(x))$ onto the set $R_1(x)$.

This reduction closes the class DelayP as stated in the following lemma.

Lemma 9. *Let R_1 and R_2 be two polynomially bounded relations. Suppose that there exists a polynomial enumeration reduction from $\text{ENUM}(R_1)$ to $\text{ENUM}(R_2)$. If $\text{ENUM}(R_2)$ belongs to DelayP, then $\text{ENUM}(R_1)$ belongs to DelayP.*

Proof. Suppose (σ, τ) is a polynomial reduction from $\text{ENUM}(R_1)$ to $\text{ENUM}(R_2)$, x is an instance of $\text{ENUM}(R_1)$ and \mathcal{A} is a polynomial delay algorithm for $\text{ENUM}(R_2)$. The following algorithm, solves $\text{ENUM}(R_1)$ with polynomial delay:

Algorithm 4: Polynomial algorithm for $\text{ENUM}(R_1)$

Input: $x \in \Sigma^*$ an instance of $\text{ENUM}(R_1)$

- 1 Calculate $\sigma(x)$
- 2 Simulate \mathcal{A} on $\sigma(x)$
- 3 **for** each solution $y \in R_2(\sigma(x))$ produced by \mathcal{A} **do**
- 4 └ Output $\tau(\sigma(x), y)$

Let us analyse the first delay of this algorithm on the input x . We compute first $\sigma(x)$, \mathcal{A} produces y_1 the first solution on $\sigma(x)$ and finally we compute $\tau(\sigma(x), y_1)$. Since by definition σ and τ are polynomial computable and since \mathcal{A} has polynomial delay, this first delay is polynomial. For the production of the following solutions, each time we have to let \mathcal{A} produce y_i its next solution on $\sigma(x)$ and to compute $\tau(\sigma(x), y_i)$. This is also polynomial. \square

This notion of reduction is very strict and can be compared to the notion of parsimonious reduction in the framework of counting, see Valiant's work [97]. Such a reduction establishes a one-to-one correspondence between the sets of solutions. This requirement can be relaxed as it was done for instance by Mary and Rospocher in their respective thesis [79, 88] and by Creignou et al. in [26].

Polynomial enumeration reductions between problems will be used several times in Chapter 4. In Chapter 3, where we are not interested in polynomial delay but in linear delay, we will need a tighter notion of reduction, which preserves the existence of a linear delay algorithm.

Definition 10 (linear enumeration reduction). *Let $\text{ENUM}(R_1)$ and $\text{ENUM}(R_2)$ be two enumeration problems. A linear enumeration reduction from $\text{ENUM}(R_1)$ to $\text{ENUM}(R_2)$ is a pair of mappings (σ, τ) with $\sigma : \Sigma^* \rightarrow \Sigma^*$ and $\tau : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ which satisfies:*

- σ is computable in polynomial time;
- $\tau(x, y)$ is computable in linear time in the size of y ;
- for every $x \in \Sigma^*$, the mapping $y \mapsto \tau(\sigma(x), y)$ is one-to-one correspondence from the set $R_2(\sigma(x))$ onto the set $R_1(x)$ and $|y|$ is bounded by a linear function in the size of $\tau(\sigma(x), y)$.

Lemma 11. *Let R_1 and R_2 be two polynomially bounded relations. Suppose that there exists a linear enumeration reduction from $\text{ENUM}(R_1)$ to $\text{ENUM}(R_2)$. If there exists a linear delay algorithm for $\text{ENUM}(R_2)$, then there exists a linear delay algorithm for $\text{ENUM}(R_1)$.*

Proof. We use the same algorithm as in the proof of Lemma 9. Only the complexity analysis of the i -th delay for $i \geq 2$ differs. After the production of the $(i-1)$ -th solution, \mathcal{A} produces y_i its i -th solution on $\sigma(x)$ and then we compute $\tau(\sigma(x), y_i)$. The first step requires a time bounded by a linear function in $|y_{i-1}| + |y_i|$ since \mathcal{A} has linear delay, and the second one a time bounded by a linear function in $|y_i|$ by assumption on τ (see Definition 10). So the i -th delay is bounded by a linear function in $|y_{i-1}| + |y_i|$. By definition of a linear enumeration reduction, see Definition 10, $|y_{i-1}| + |y_i|$ is bounded by a linear function in $|\tau(\sigma(x), y_{i-1})| + |\tau(\sigma(x), y_i)|$. Hence the i -th delay is bounded by a linear function in the size of the $(i-1)$ -th and i -th solutions, thus proving that the algorithm has linear delay. \square

In Chapter 3, besides this notion of linear enumeration reduction, we will use another type of reductions among enumeration problems, reminiscent of classical Turing reductions. Roughly speaking such a Turing reduction will allow us to solve $\text{ENUM}(R_1)$ in using $\text{ENUM}(R_2)$ has an enumeration oracle.

2.3 Hardness results

Intractability results are rare in enumeration theory. Intractability of enumeration is mostly proved by showing intractability of a related decision problem rather than directly proving lower bounds by relating one enumeration problem to the other. The propositions below show how the complexity of enumeration relates to the complexity of associated tasks. These results are now part of the folklore. Here we will lean on the work of Schmidt [89]. In all the section R denotes a polynomially bounded binary relation over a finite alphabet.

Proposition 12. *If $\text{ENUM}(\mathcal{R}) \in \text{TotalP}$, then $\text{DECIDE}(\mathcal{R}) \in \text{P}$*

Proof. We use a TotalP-algorithm \mathcal{A} for $\text{ENUM}(\mathcal{R})$ to decide whether a given instance has a solution. Let x be an instance of $\text{ENUM}(\mathcal{R})$ and let p be a polynomial such that the total running time of \mathcal{A} is exactly $p(|x|, |R(x)|)$, see Definition 2 of Section 2.1. The decision algorithm for $\text{DECIDE}(\mathcal{R})$ is the following:

Algorithm 5: Polynomial algorithm for $\text{DECIDE}(\mathcal{R})$

Input: $x \in \Sigma^*$

```

1 Simulate  $\mathcal{A}$  on  $x$  for  $p(|x|, 0)$  steps
2 if  $\mathcal{A}$  has terminated then
3   | Return "no"
4 else
5   | Return "yes"

```

We know that on input x , algorithm \mathcal{A} stops after exactly $p(|x|, 0)$ steps if and only if $R(x) = \emptyset$. Hence, the algorithm says "yes" if and only if $R(x) \neq \emptyset$. Obviously the running time of this decision algorithm is $O(p(|x|, 0))$. □

An analogous statement involving the associated computational task $\text{ALLSOL}(\mathcal{R})$ can be proven similarly.

Proposition 13. *If $\text{ENUM}(\mathcal{R}) \in \text{TotalP}$, then $\text{ALLSOL}(\mathcal{R}) \in \text{P}$.*

Proof. We use a TotalP-algorithm \mathcal{A} for $\text{ENUM}(\mathcal{R})$ to decide whether given an instance and a solution set S there exists $y \in R(x) \setminus S$. Let x be an instance of $\text{ENUM}(\mathcal{R})$ and let p be a polynomial such that the total running time of \mathcal{A} is exactly $p(|x|, |R(x)|)$. The decision algorithm for $\text{ALLSOL}(\mathcal{R})$ is the following:

Algorithm 6: Polynomial algorithm for $\text{ALLSOL}(\mathcal{R})$

Input: $x \in \Sigma^*, S \subseteq R(x)$

```

1 Simulate  $\mathcal{A}$  on  $x$  for  $p(|x|, |S|)$  steps
2 if  $\mathcal{A}$  has terminated then
3   | Return "no"
4 else
5   | Return "yes"

```

We know that on input x , algorithm \mathcal{A} stops after exactly $p(|x|, |S|)$ steps if and only if $R(x) = S$. Hence, the algorithm says "yes" if and only if $R(x) \neq S$. Obviously the running time of the decision algorithm below is $O(p(|x|, |S|))$. □

Therefore, under the assumption $\text{P} \neq \text{NP}$, in order to prove that $\text{ENUM}(\mathcal{R})$ does not belong to TotalP (and *a fortiori* does not belong to either IncP or DelayP), one can prove P-hardness of either $\text{DECIDE}(\mathcal{R})$ or $\text{ALLSOL}(\mathcal{R})$. The enumeration problem associated with the boolean satisfiability problem SAT is an example of such a "hard" enumeration problem since the decision problem SAT is NP-complete.

Interestingly, it turns out that the enumeration complexity class IncP is characterized by the complexity of the search problem ANOTHERSOL.

Proposition 14. $\text{ENUM}(\mathbf{R}) \in \text{IncP}$ if and only if $\text{ANOTHERSOL}(\mathbf{R}) \in \text{FP}^1$.

Proof. Suppose that $\text{ENUM}(\mathbf{R}) \in \text{IncP}$ and consider an IncP-algorithm \mathcal{A} for $\text{ENUM}(\mathbf{R})$ and p its associated polynomial. Observe that on input x , algorithm \mathcal{A} computes at least k solutions in $\sum_{i=0}^k (p(|x|, i))$ steps for any $k \leq |R(x)|$. We can use \mathcal{A} for solving $\text{ANOTHERSOL}(\mathbf{R})$ in polynomial time:

Algorithm 7: Polynomial algorithm for $\text{ANOTHERSOL}(\mathbf{R})$

Input: $x \in \Sigma^*$, $S \subseteq R(x)$

- 1 Simulate \mathcal{A} on x for at most $\sum_{i=0}^{|S|+1} (p(|x|, i))$ steps
- 2 **if** \mathcal{A} has output less than $|S| + 1$ solutions **then**
- 3 Return "NIL"
- 4 **else**
- 5 Search a solution $y \in A(x) \setminus S$
- 6 Output y

We can assume $\sum_{i=0}^{|S|+1} (p(|x|, i))$ to be bounded by $(|S| + 2) \times p(|x|, |S| + 1)$, thus to be polynomial in $|x|$ and $|S|$.

Now, if $\text{ANOTHERSOL}(\mathbf{R}) \in \text{FP}$ there is a straightforward algorithm to show that $\text{ENUM}(\mathbf{R}) \in \text{IncP}$. We denote by \mathcal{A} the polynomial algorithm that solves $\text{ANOTHERSOL}(\mathbf{R})$.

Algorithm 8: Incremental algorithm for $\text{ENUM}(\mathbf{R})$

Input: $x \in \Sigma^*$

- 1 $S = \emptyset$
- 2 **while** $\mathcal{A}(x, S) \neq \text{NIL}$ **do**
- 3 Output $\mathcal{A}(x, S)$
- 4 $S = S + \mathcal{A}(x, S)$

The algorithm \mathcal{A} being polynomial, the delay of the enumeration algorithm above is polynomial in the output x and the number of solutions already generated $|S|$. □

Therefore, under the assumption $\text{P} \neq \text{NP}$, in order to prove that $\text{ENUM}(\mathbf{R})$ does not belong to IncP (and *a fortiori* does not belong to DelayP), one can prove hardness of $\text{ANOTHERSOL}(\mathbf{R})$.

We saw that intractability of an enumeration enumeration can be proved in showing the difficulty of some associated computational tasks. There is, however, another method to prove that an enumeration problem is hard: showing that it is as difficult as a problem known to be difficult. For example, a way to prove hardness leans on the generation problem of minimal transversals of a hypergraph.

A *hypergraph* $\mathcal{H} = (V(\mathcal{H}), \mathcal{E}(\mathcal{H}))$ is a generalisation of a graph where $V(\mathcal{H})$ denotes the set of vertices and $\mathcal{E} \subseteq 2^V$ is a collection of sets of vertices called *hyperedges*. Thus, a graph is a particular hypergraph where all hyperedges are of size two. A *transversal* of a hypergraph is a set of vertices that intersects all

¹FP is the set of search problems that are calculable in polynomial time.

hyperedges. Similarly as for the vertex set cover problem for graphs, for hypergraphs we are interested in minimal transversals (i.e., inclusion minimal hitting sets) and the enumeration problem associated is certainly one of the most studied enumeration problems, thanks to its connections with many areas.

ENUM_MINIMAL-TRANSVERSALS

Input : a hypergraph $\mathcal{H} = (V(\mathcal{H}), \mathcal{E}(\mathcal{H}))$

Output : the set of all minimal transversals of \mathcal{H}

Computing or recognizing the minimal transversals of a hypergraph is a frequent problem in practice, it has important applications in data-mining and machine learning [13,57], logic and artificial intelligence [31]. Despite the interest this problem has attracted, it is a longstanding open question whether the enumeration of minimal transversals of a hypergraph is in the class TotalP. Hence, proving that an enumeration problem is equivalent to the enumeration of minimal transversals gives an indication of its hardness.

In the next chapter, we will focus on the enumeration of minimal dominating sets and maximal irredundant sets. On the one hand, the enumeration of minimal dominating sets is an example of enumeration problem that has been proved to be equivalent to the enumeration of minimal transversals by Kante *et al.* [65]. On the other hand, Boros and Makino [14] proved that it is not possible to enumerate maximal irredundant sets with incremental delay unless $P = NP$ in showing that the corresponding ANOTHERSOL problem is NP-hard (Proposition 14). Therefore, enumerating minimal dominating sets, as well as enumerating maximal irredundant sets, are supposed to be difficult enumeration problems in general. For this reason, in this thesis when studying these problems, we will restrict our attention to special graph classes.

Efficient enumeration of vertex set properties in graphs

Contents

3.1	Introduction	21
3.2	Preliminaries	23
3.2.1	Definitions and properties on graphs	23
3.2.2	Vertex set properties and constraints on graphs	26
3.2.3	Paths in a directed acyclic graph	27
3.3	Local properties and enumeration for linearly ordered graphs	30
3.4	MDS, MCDS and MIR in interval graphs	33
3.4.1	Interval graphs and locally definable basic properties	33
3.4.2	Interval graphs and minimal dominating sets	35
3.4.3	Interval graphs and minimal connected dominating sets	37
3.4.4	Interval graphs and maximal irredundant sets	39
3.5	MDS, MCDS and MIR in permutation graphs	41
3.5.1	Permutation graphs and locally definable basic properties	41
3.5.2	Permutation graphs and minimal dominating sets	47
3.5.3	Permutation graphs and minimal connected dominating sets	48
3.5.4	Permutation graphs and maximal irredundant sets	50
3.6	Local properties and enumeration for cyclically ordered graphs	53
3.7	MDS and MIR in circular-arc and circular-permutation graphs	57
3.7.1	Enumeration in circular-arc graphs	57
3.7.2	Enumeration in circular-permutation graphs	60
3.8	Conclusion	61

3.1 Introduction

A famous example of enumeration problem in graph theory is that of enumerating the minimal dominating sets of a graph [23,24,38]. This problem has been solved with linear delay for the interval and permutation graph classes, by M.M.Kanté *et al.* in [66]. Mary observed in his dissertation [79] that such a reduction

seems to be possible when the considered graph property presents a local character. We propose in this thesis to build on this observation in order to put forward a general method for finding linear delay algorithms for the enumeration of vertex set properties in graphs.

We consider graphs in which there is a natural order on the vertices, namely linearly ordered graphs and cyclically ordered graphs. We say that a property is locally definable with respect to a linear or a cyclic order on the vertices of the graph, when one can decide if a subset of vertices respects the property by scanning its ordered list of vertices through a sliding window of fixed size and verifying that each scanned tuple satisfies some constraints. Our main result is that such a locally definable vertex set property can be enumerated with linear delay after a polynomial pre-processing. This result is obtained by reducing the enumeration of vertex sets satisfying such a locally definable property to the enumeration of paths in a directed acyclic graph (DAG). In the case of linearly ordered graphs the reduction establishes a one-to-one correspondence between the vertex sets to enumerate and the paths in a DAG. Interestingly, in the case of cyclically ordered graphs we use a Turing reduction, which requires a polynomial number of calls to the procedure that enumerates the paths in a DAG.

We illustrate the interest of this method by applying it to variants of the minimal dominating set problem, namely, the minimal connected dominating set and the maximal irredundant set problem. These vertex set properties are well studied in graph theory, especially as optimisation problems. Both computing a cardinality minimum connected dominating set and computing a cardinality maximum irredundant set are NP-hard [33, 47]. Input sensitive algorithms have been proposed for enumerating minimal connected dominating sets and maximal irredundant sets on special classes of graphs (see e.g. [53, 54]). Boros and Makino [14] proved that it is not possible to enumerate maximal irredundant sets with polynomial delay (and even incremental delay) unless $P = NP$. Here, we restrict the study of the enumeration of minimal (connected) dominating sets and maximal irredundant sets to the following graph classes: interval and permutation graphs for linearly ordered graphs and circular-arc and circular-permutation graphs for cyclically ordered graphs. These graph classes have numerous applications (see e.g. [9, 55, 100]), and are classes in which many NP-complete problems can be solved efficiently (see [9] for some examples).

In applying our method we prove that minimal connected dominating sets can be enumerated with linear delay on interval graphs and permutation graphs. As a corollary we obtain that one can compute a minimum connected dominating set on such graphs in polynomial time. We also prove that maximal irredundant sets can be enumerated with linear delay on interval graphs, permutation graphs, circular-arc and circular-permutation graphs. Moreover on these graph classes, computing a maximum irredundant set can be done in polynomial time.

The minimal dominating set and the maximal irredundant set problems being at the core of this chapter, we will also take the time to study the relation between these two problems. In fact, on one hand the literature contains extensive studies of variations of the domination problem [59, 60, 98], but and the other hand the results on the relation between the set of minimal dominating sets and the set of maximal irredundant sets are less abundant. Hence, we propose a characterization, in terms of forbidden subgraphs, of the graphs that respect for each of their induced subgraph the property that the set of minimal dominating sets (MDS) and the set of maximal irredundant sets (MIR) correspond.

Section 3.2 is devoted to some general preliminaries and to the study of the graphs for which the hereditary property $MIR = MDS$ holds. Section 3.3 states the main result for linearly ordered graphs, roughly speaking: the collections of sets of vertices that can be locally defined with respect to a linear order on the vertices of the graph are enumerable with linear delay. In the next two sections, we apply this method to concrete problems: the minimal connected dominating set and the maximal irredundant set problems are addressed in the case of interval graphs in Section 3.4 and in the case of permutation graphs in Section 3.5. Finally, in Section 3.6 we state the second result for cyclically ordered graphs: the

collections of sets of vertices that can be locally defined with respect to a cyclic order on the vertices of the graph are enumerable with linear delay. This result is then illustrated in Section 3.7 with the enumeration of maximal irredundant sets in circular-arc and circular-permutation graphs.

3.2 Preliminaries

After a reminder of some definitions and properties on graphs, we state a characterization of the graphs that respect for each of their induced subgraph the property that the set of minimal dominating sets and the set of maximal irredundant sets correspond. Then, we introduce the notion of vertex set property and we recall the algorithm that enumerates with linear delay the set of paths of a directed acyclic graph.

3.2.1 Definitions and properties on graphs

All graphs considered in this chapter are finite. If $G = (V, E)$ is a graph and X is a subset of V , we denote by G_X the subgraph of G induced by X . The *set of neighbours* of a vertex x is defined by $N(x) = \{y \in V \mid \{x, y\} \in E\}$ while the *closed set of neighbours* is $N[x] = N(x) \cup \{x\}$. We set $N[X] = \bigcup_{x \in X} N[x]$ and $N(X) = N[X] \setminus X$ (for the sake of readability we often write $N(x_1, \dots, x_k)$ instead of $N(\{x_1, \dots, x_k\})$, and the same for $N([x_1, \dots, x_k])$). A neighbour y of some $x \in X$ is a *private neighbour with respect to X* if y has no other neighbour in X . In other terms, the set $P_X(x)$ of private neighbours of x with respect to X fulfills $P_X(x) = N[x] \setminus N[X \setminus x]$. For convenience, we will omit the expression “with respect to X ” when X is clear from the context. For any set of vertices X and any $x \in X$ we denote by $\deg_X(x)$ the degree of x in G_X , that is the integer $|N(x) \cap X|$. Besides, we set $\deg(X) = \max\{\deg_X(x), x \in X\}$.

Given a graph $G = (V, E)$, we call a subset $X \subseteq V$:

- a *dominating set* (DOM) if each $x \in V \setminus X$ has a neighbour in X (see Figure 3.1);
- an *irredundant set* (IRR) if each $x \in X$ has a private neighbour with respect to X (see Figure 3.4);
- a *minimal dominating set* (MDS) if it is irredundant and dominating (see Figure 3.2);
- a *minimal connected dominating set* (MCDS) if it is a connected dominating set and for any $v \in X$, $X \setminus \{v\}$ is either not connected or not dominating (see Figure 3.3);
- a *maximal irredundant set* (MIR) if it is an irredundant set maximal for inclusion (see Figure 3.5).

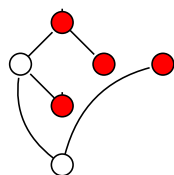


Figure 3.1: DOM

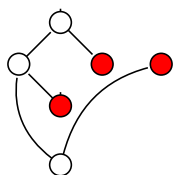


Figure 3.2: MDS

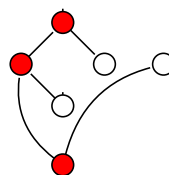


Figure 3.3: MCDS

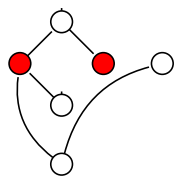


Figure 3.4: IRR

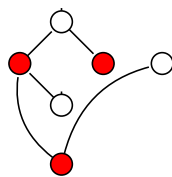


Figure 3.5: MIR

For any graph G , every minimal dominating set X is a maximal irredundant set. This follows naturally from the definition of minimal dominating set. In fact, let X be a minimal dominating set, because of the minimality, X is irredundant. If it is not maximal, there exists Y an irredundant set such that $X \subset Y$ and if we take $u \in Y$ with $u \notin X$, we have that u has a private neighbour v with respect to Y , i.e., $v \in N[u]$ and $v \notin N[Y \setminus \{u\}]$. The vertex v is not dominated by X and we have here our contradiction.


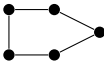
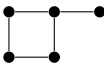
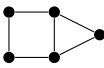
Conversely it is not the case that every maximal irredundant set is a minimal dominating set. See Figure 3.6 below for example.



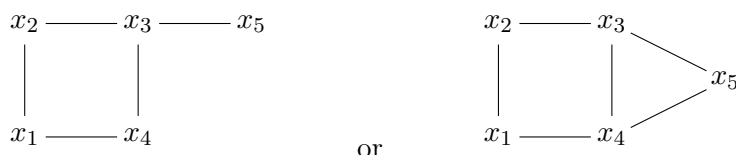
Figure 3.6: A MIR that is not a MDS

Nevertheless there are graphs for which these two notions coincide, that is for which $MIR = MDS$. It is the case for example for split graphs [65] and we can pretty quickly see that it works for cographs too. A natural question that follows is that of identifying all the graphs for which we have this property on every induced subgraph. Such identification has also relevant consequences in enumeration theory. A lot of work has been done to solve the problem of enumerating all minimal dominating sets on particular graph classes [23, 24, 64, 66]. Results for the enumeration of maximal irredundant sets are however less abundant. Identifying graphs with hereditary property $MIR = MDS$, enables us not only to expand our understanding of the relation between minimal dominating sets and maximal irredundant sets but gives us also new results on the enumeration of maximal irredundant sets. In the following, we propose a characterization of those graphs with hereditary property $MDS = MIR$ in terms of forbidden induced subgraphs. We call this new class of graphs, the class of *MDS-MIR perfect graphs*.

Theorem 15. *A graph G belongs to the class of MDS-MIR perfect graphs if and only if G does not contain any induced subgraph of the form*

1.  (P_5)
2.  (C_5)
3.  (4-pan)
4.  (*House graph*)

Proof. Let G be a MDS-MIR perfect graph, and let us show that G does not contain any induced subgraph of the form P_5 , C_5 , 4 -pan or *House graph*. Suppose by contradiction that there exists G' an induced subgraph of the form $P_5 = \{x_1, x_2, x_3, x_4, x_5\}$. Then $\{x_2, x_3\}$ is a maximal irredundant set but x_5 is not dominated. We have a contradiction with the fact that G' respects $\text{MIR} = \text{MDS}$. In the same way, if G' is an induced subgraph of the form $C_5 = \{x_1, x_2, x_3, x_4, x_5, x_6 = x_1\}$, then $\{x_5, x_1\}$ is a maximal irredundant set but x_3 is not dominated. Finally, if G' is of the form



then $\{x_1, x_2\}$ are maximal irredundant sets that are not dominating sets.

Now, let G be a graph without induced subgraph of the form P_5 , C_5 , 4 -pan or *House pan* and let us show that G is a MDS-MIR perfect graph. By contradiction suppose that G is not MDS-MIR perfect, i.e., there exists G' an induced subgraph of G such that G' has a maximal irredundant set that is not a minimal dominating set. The claim below gives rise to the contradiction that completes the proof.

Claim 16. *If G is a graph and X a maximal irredundant set that is not a minimal dominating set, then there exists an induced subgraph of G of the form P_5 or C_5 or 4 -pan or *House graph*.*

Let us explain this claim. First observe that if G is a graph and X a maximal irredundant set of G , then either X is not a dominating set or it is a minimal dominating set. Consider a set of vertices X that is maximal irredundant of G but not a minimal dominating set. Hence, X is not dominating and so there exists a vertex x that is not dominated, i.e., $N[x] \cap X = \emptyset$. Let us consider $X \cup \{x\}$. Because of the maximality of X , $X \cup \{x\}$ is not irredundant. The vertex x cannot be isolated (X would not be maximal) and is its own private neighbour with respect to $X \cup \{x\}$ because $N[x] \cap X = \emptyset$. We conclude that there exists $z \in X$ such that $P_X(z) \subseteq N(x)$ (otherwise $X \cup \{x\}$ would be irredundant). Let us take $y \in P_X(z)$. Since $P_X(z) \subseteq N(x)$ and $N[x] \cap X = \emptyset$, we have $y \notin X$. The vertex z cannot be its own private neighbour (since $N[z] \cap X \neq \emptyset$), and so it has to be connected to some other element z' of X . The vertex z' too has a private neighbour y' with respect to X . Since z' is a neighbour of $z \in X$, $y' \notin X$. This is why the minimal configuration required so that X is a maximal irredundant set and not a minimal dominating set is the following:

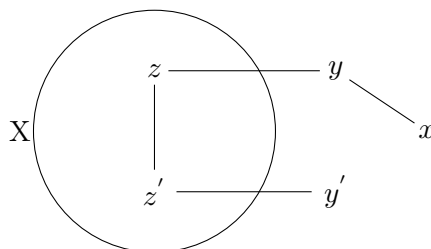


Figure 3.7: Minimal configuration P_5

From the minimal configuration illustrated in Figure 3.7, it is possible to obtain the four induced subgraphs of Figure 3.8 below:

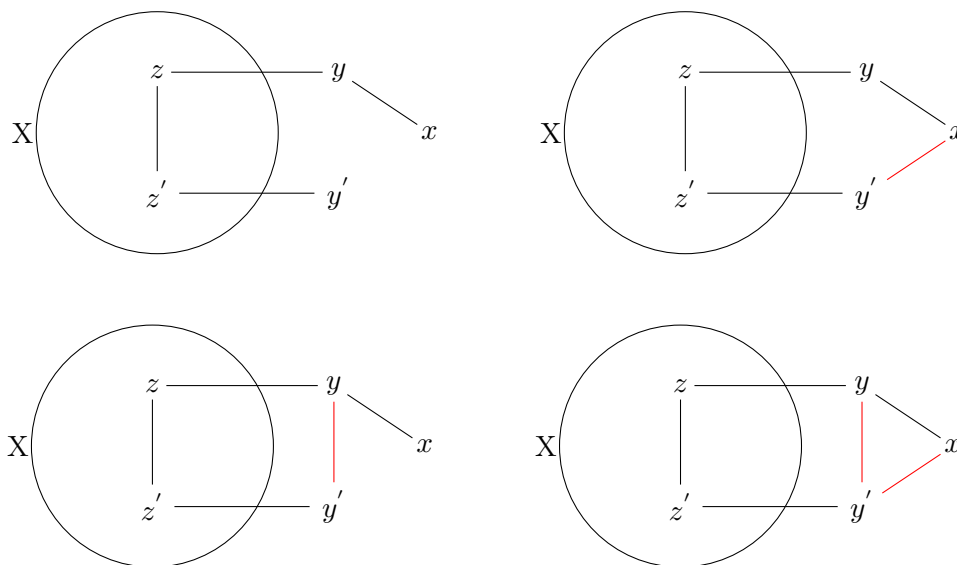


Figure 3.8: Forbidden induced subgraphs P_5 , C_5 , 4-pan and House graph

In fact, it is not possible to add to Figure 3.7 the edge $\{z, x\}$ or $\{z', x\}$ because X does not dominate x . It is also not possible to add the edge $\{z', y\}$ or $\{z, y'\}$ because X is irredundant. □

The minimal dominating set property and the maximal irredundant set property are examples of vertex set properties that are locally definable on some classes of graphs, and thus that can be enumerated by reduction to the enumeration of paths in a DAG. Let us give the definitions and basic results concerning vertex set properties and enumeration of paths in a DAG.

3.2.2 Vertex set properties and constraints on graphs

Let us define formally the notions of vertex set properties and vertex constraints. These notions will enable us to construct a framework that allows for linear delay enumeration for a very broad class of graphs and problems.

Definition 17. A vertex set property is a recursive function \mathcal{P} mapping each graph $G = (V, E)$ to a collection of subsets of vertices, $\mathcal{P}_G \subseteq 2^V$. We say that \mathcal{P} is a polynomial vertex set property if given G and X , one can decide whether $X \in \mathcal{P}_G$ in polynomial time.

Example. The dominating set property is the application that maps each graph to its collection of dominating sets. The irredundant set property is similarly defined. Both these vertex set properties are polynomial. ◁

Definition 18. Let \mathcal{G} be a set of graphs. A vertex constraint of arity k on \mathcal{G} is a recursive function ϕ that maps each $G = (V, E)$ of \mathcal{G} to a set $\phi_G \subseteq V^k$. We say that such a constraint is decidable in polynomial time if given G and a tuple of vertices (a_1, \dots, a_k) , one can decide whether $\phi_G(a_1, \dots, a_k)$ holds in polynomial time.

An *ordered graph* (G, \preceq) is a graph G whose set of vertices is equipped with a relation order \preceq . Given a set of graphs \mathcal{G} , a *uniform order* on \mathcal{G} is a recursive function \preceq mapping each $G \in \mathcal{G}$ to an order of its vertices. Unless it leads to confusion, we will denote by the same symbol \preceq the above function and the order it associates with a given graph. A set of graphs \mathcal{G} equipped with a uniform order \preceq is said uniformly ordered and denoted by (\mathcal{G}, \preceq) .

Definition 19. Let (G, \preceq) be an ordered graph and $X \subseteq V(G)$. A k -window of X w.r.t. \preceq is a k -tuple $(a_1, \dots, a_k) \in X^k$ such that for each $i < k$, a_{i+1} is the \preceq -successor of a_i in X .

In the following by abuse of notation we write $X = \{x_1 \prec x_2 \prec \dots \prec x_q\}$ to denote a set $X = \{x_1, x_2, \dots, x_q\}$ with $x_1 \prec x_2 \prec \dots \prec x_q$.

Example. If $V(G)$ is ordered and if $X = \{x_1 \prec x_2 \prec \dots \prec x_q\}$, then the tuple (x_3, x_4) is a 2-window and $(x_{q-3}, x_{q-2}, x_{q-1}, x_q)$ is a 4-window of X w.r.t. \preceq . \triangleleft

When \preceq is clear from the context, we will talk about “ k -windows of X ” and we will denote the set of k -windows of X w.r.t. \preceq by $\mathbb{W}^k(X)$.

In this chapter, we are interested in the enumeration of vertex set properties \mathcal{P} , that is in the following problem:

<p style="text-align: center;">ENUM_ \mathcal{P}</p> <p><i>Input</i> : a graph G</p> <p><i>Output</i> : all $X \subseteq V(G)$ s.t. $X \in \mathcal{P}_G$</p>
--

The method we will present to solve those problems consists in a reduction to the enumeration of paths in a directed acyclic graph, which is known to be feasible with linear delay. Let us recall this result.

3.2.3 Paths in a directed acyclic graph

Given a DAG G and two subsets of vertices S and T , it is well-known that paths in G from S to T can be enumerated with linear delay (see e.g., [66]).

Recall that this means that there exists an algorithm \mathcal{A} with pre-processing (and generation of the first solution) bounded by a polynomial in the size of the input and that generates all paths of G from S to T such that the maximal time elapsed between the production of two consecutive solutions is bounded in the sum of the sizes of these two solutions.

<p style="text-align: center;">ENUM_PATHSDAG</p> <p><i>Input</i> : a DAG $G = (V, E)$ and $S, T \subseteq V$</p> <p><i>Output</i> : all paths of G from S to T</p>
--

The pre-processing consists in removing from the graph G all vertices from which it is impossible to reach T . It is sufficient for the enumeration afterward because the algorithm solving ENUM_PATHSDAG starts from S , and thus avoids to visit the vertices which lead to false tracks. This pre-processing step is referred to as CLEAR DAG.

Algorithm 9: CLEARDAg

Input: a DAG $G = (V, A)$, $S, T \subseteq V$
Output: G without the vertices from which T is not reachable

```

1 Let  $(x_1, \dots, x_n)$  be a topological ordering of  $G$ 
2 Mark all vertices in  $T$ 
3 for  $i = n - 1$  to 1 do
4   for each outgoing neighbour  $y$  of  $x_i$  do
5     if  $y$  is marked then
6       mark  $x_i$ 
7 Delete from  $G$  all vertices that are not marked
8 Return  $G$ 

```

Observe that the complexity of this function CLEARDAg is linear in the size of G . In fact, it is known that finding a topological ordering can be done in $O(m + n)$. Furthermore, lines 4-5 consists for each vertices to go through its outgoing arcs, this means that we go through the m edges exactly once. Hence, lines 4-5 take a time in $O(m)$.

Once the DAG has been cleared from the vertices that do not lead to T , a "start vertex" s and a "target vertex" t are added to the DAG and the enumeration is then obtained by a depth-first exploration of the remaining graph.

Algorithm 10: ENUM_PATHSDAG

Input: a DAG $G = (V, A)$, $S, T \subseteq V$
Output: list all paths from S to T

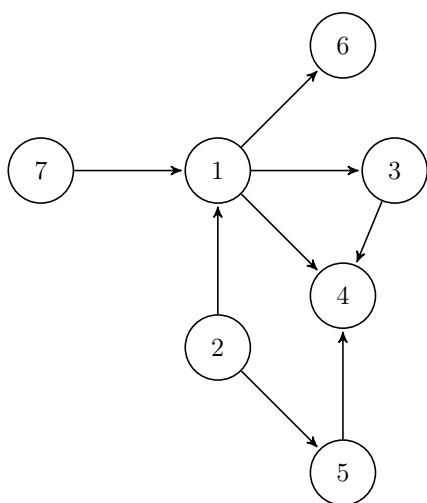
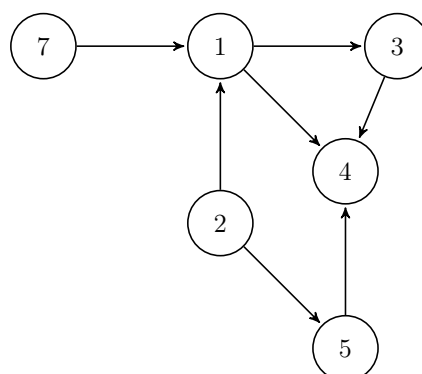
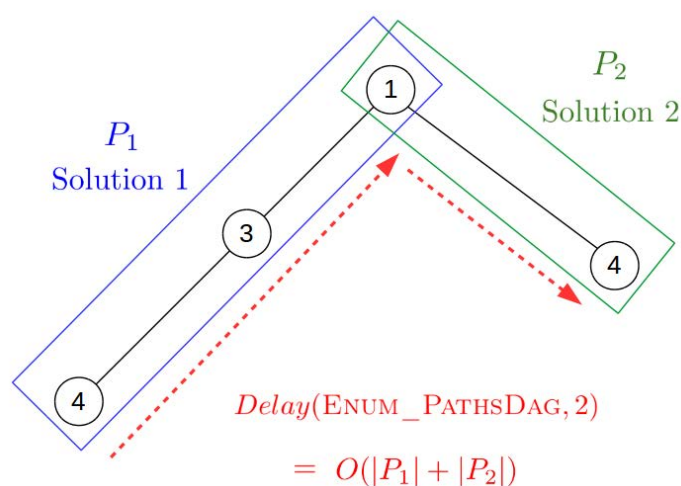
```

1  $G' \leftarrow \text{CLEARDAg}(G, S, T)$ 
2 Let  $s$  and  $t$  be new vertices
3 Add the arc  $(s, x)$  to  $G'$  for all  $x \in S$ 
4 Add the arc  $(x, t)$  to  $G'$  for all  $x \in T$ 
5 Generate( $G', s, t, \emptyset$ )

6 Function Generate (a DAG  $G = (V, A)$ , a vertex  $x$ , a sink  $t$  and  $D \subseteq V$ ):
7   if  $x = t$  then
8     output( $D$ )
9   else
10    for each outgoing neighbour  $y$  of  $x$  do
11      Generate( $G, y, t, D \cup \{y\}$ )

```

ENUM_PATHSDAG has a pre-processing (procedure CLEARDAg) in $O(m+n)$ and a delay in $O(|P_{i-1}| + |P_i|)$, where P_i stands for the i -th path of the DAG, see Figures 3.9, 3.10 and 3.11.

Figure 3.9: G initial graphFigure 3.10: $G' = \text{CLEAR DAG}(G, \{1\}, \{4\})$ Figure 3.11: $\text{Generate}(G', 1, 4, \emptyset)$

This leads to the theorem below, which will play a key role in the sequel.

Theorem 20 (folklore). *Given a directed acyclic graph G and two subsets S and T of vertices of G , all paths in G from vertices in S to vertices in T can be enumerated with linear delay after a linear time pre-processing.*

It is possible to be more precise and to solve ENUM_PATHSDAG in such a way that the delay between the production of two consecutive solutions P_{i-1} and P_i is linear in $|P_i|$. To such a purpose, the recursivity needs to be simulated with the help of adapted data structures in such a way that when we have reached a solution, we have a direct access in the exploration tree to the next node for which the exploration is not yet complete.

3.3 Local properties and enumeration for linearly ordered graphs

In [66], Kanté et al. reduced the problem of enumerating the minimal dominating sets of an interval (resp. permutation) graph to that of enumerating paths of a directed acyclic graph, thus obtaining efficient enumeration algorithms. In this section we generalize this method by specifying properties whose enumeration can be reduced to enumerating paths in a DAG.

We will focus on vertex set properties, which can be “locally defined” with respect to a linear order on the vertices of the graph. Roughly speaking, a vertex set property \mathcal{P} is local with respect to some class \mathcal{G} of linearly ordered graphs if for any $G \in \mathcal{G}$ and any $X \subseteq V(G)$, one can decide whether $X \in \mathcal{P}_G$ by scanning its ordered list of vertices through a sliding window of fixed size and verifying that each scanned tuple satisfies a predefined constraint. In general, a window carries information on the elements the window covers. A special care has to be brought to extremal windows, which have also to carry information on external vertices that are at the left or the right of the window. Let us formalize this definition.

Definition 21. Let (\mathcal{G}, \preceq) be a uniformly linearly ordered set of graphs. A vertex set property \mathcal{P} is locally definable on (\mathcal{G}, \preceq) if there exist an integer $k > 1$ and three vertex constraints ϕ , ϕ^{\min} and ϕ^{\max} of respective arities k , $k-1$ and $k-1$, such that for any graph $G \in \mathcal{G}$ and any subset $X = \{x_1 \prec x_2 \prec \dots \prec x_q\} \subseteq V(G)$ of cardinality $\geq k$, we have:

$$X \in \mathcal{P}_G \text{ iff } \left\{ \begin{array}{l} \forall (x_i, \dots, x_{i+k-1}) \in \mathbb{W}^k(X) : \phi_G(x_i, \dots, x_{i+k-1}) \\ \text{and } \phi_G^{\min}(x_1, \dots, x_{k-1}) \\ \text{and } \phi_G^{\max}(x_{q-k+1}, \dots, x_q) \end{array} \right\} \quad (3.1)$$

Furthermore, if \mathcal{P} is a polynomial vertex set property and if the three constraints ϕ , ϕ^{\min} and ϕ^{\max} are decidable in polynomial time, then \mathcal{P} is said to be locally definable in polynomial time on (\mathcal{G}, \preceq) .

Note that in the above definition, the integer k is a constant independent of the input graph G .

Roughly speaking, when $X = \{x_1 \prec x_2 \prec \dots \prec x_q\}$, a k -window of X will be used to check properties on the interval $[x_1, x_q]$. Nevertheless, there are in G vertices that are either smaller than x_1 or larger than x_q . For these vertices we will use extremal windows of smaller size. Figure 3.12 below illustrates this operation, for $k = 3$ and $V = \{v_1, \dots, v_n\}$.

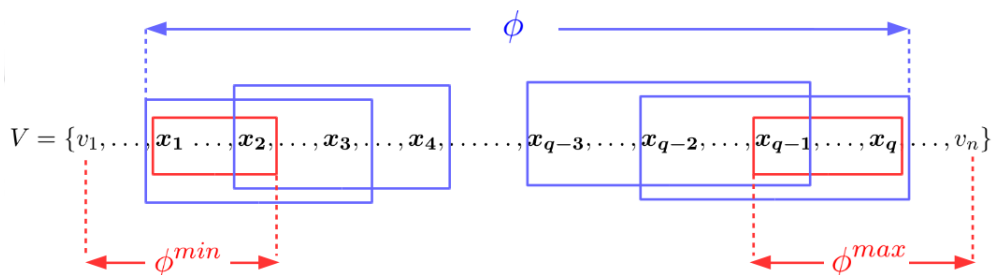


Figure 3.12: Locally definable properties through 3-windows in linearly ordered graphs

Our main result states that such locally definable properties are efficiently enumerable.

Theorem 22. *Let (\mathcal{G}, \preceq) be a uniformly linearly ordered set of graphs and \mathcal{P} be a vertex set property. If \mathcal{P} is locally definable in polynomial time on (\mathcal{G}, \preceq) , then $\text{ENUM_}\mathcal{P}$ is enumerable with linear delay.*

Proof. We prove this result by reduction to the problem ENUM_PATHSDAG . Let us consider a vertex set property \mathcal{P} that is locally definable on (\mathcal{G}, \preceq) in polynomial time. Let k be the integer and ϕ , ϕ^{\min} and ϕ^{\max} be the three vertex constraints associated to \mathcal{P} according to Def. 21. For every linearly ordered graph $G = (V, E, \preceq)$ in (\mathcal{G}, \preceq) , we define a new graph $G_\Delta = (V_\Delta, E_\Delta)$ as follows: vertices of G_Δ are tuples $(a_1, \dots, a_{k-1}) \in V^{k-1}$ such that (a_1, \dots, a_{k-1}) is strictly increasing w.r.t. \preceq , and a pair of vertices $((a_1, \dots, a_{k-1}), (b_1, \dots, b_{k-1}))$ is an arc in E_Δ if the following conditions hold:

$$(\Delta_1) \quad (a_2, \dots, a_{k-1}) = (b_1, \dots, b_{k-2});$$

$$(\Delta_2) \quad \phi_G(a_1, a_2, \dots, a_{k-1}, b_{k-1}) \text{ holds.}$$

Furthermore, we define the two following subsets of V_Δ :

$$(\Delta_3) \quad S_\Delta = \{(a_1, \dots, a_{k-1}) \in V_\Delta \text{ s.t. } \phi^{\min}(a_1, \dots, a_{k-1})\};$$

$$(\Delta_4) \quad T_\Delta = \{(a_1, \dots, a_{k-1}) \in V_\Delta \text{ s.t. } \phi^{\max}(a_1, \dots, a_{k-1})\}.$$

Notice that G_Δ is a DAG since any path from G_Δ is a sequence of tuples of the form $(x_1, x_2, \dots, x_{k-1})$, (x_2, x_3, \dots, x_k) , $(x_3, x_4, \dots, x_{k+1})$, \dots , $(x_p, x_{p+1}, \dots, x_{p+k-2})$ with $x_1 \prec x_2 \prec x_3 \prec \dots \prec x_p$.

Let us now prove that for any $G = (V, E)$ in \mathcal{G} , the sets $X \subseteq V(G)$ such that $X \in \mathcal{P}_G$ are in one-to-one correspondence with the paths in G_Δ from S_Δ to T_Δ . We prove that for every strictly increasing sequence x_1, \dots, x_q with $q \geq k$ in (V, \preceq) ,

$$\{x_1, \dots, x_q\} \in \mathcal{P}_G \Leftrightarrow \left(\begin{array}{c} (x_1, \dots, x_{k-1}), (x_2, \dots, x_k), \dots, (x_{q-k+2}, \dots, x_q) \\ \text{is a path from } S_\Delta \text{ to } T_\Delta \text{ in } G_\Delta. \end{array} \right)$$

\Rightarrow Let $X = \{x_1 \prec \dots \prec x_q\}$ be in \mathcal{P}_G . By definition of G_Δ , $(x_1, \dots, x_{k-1}), \dots, (x_{q-k+2}, \dots, x_q)$ are vertices of G_Δ . Every pair of two successive vertices $(x_i, x_{i+1}, \dots, x_{i+k-2})$ and $(x_{i+1}, x_{i+2}, \dots, x_{i+k-1})$ satisfies (Δ_1) . Furthermore, for i such that $1 \leq i \leq q - k + 1$, the tuple $(x_i, \dots, x_{i+k-2}, x_{i+k-1})$ is a k -window of the set $X = \{x_1, \dots, x_q\}$ and thus $\phi(x_i, \dots, x_{i+k-2}, x_{i+k-1})$ holds thanks to Def. 21. Hence, (Δ_2) is satisfied. Therefore, every pair $((x_i, \dots, x_{i+k-2}), (x_{i+1}, \dots, x_{i+k-1}))$ is an arc in G_Δ . Finally, according to Def. 21, $\phi^{\min}(x_1, \dots, x_{k-1})$ and $\phi^{\max}(x_{q-k+2}, \dots, x_q)$ hold since $x_1 = \min(X)$ and $x_q = \max(X)$. Thus, the considered sequence is a path from S_Δ to T_Δ in G_Δ .

\Leftarrow Let $(x_1, \dots, x_{k-1}), (x_2, \dots, x_k), \dots, (x_{q-k+2}, \dots, x_q)$ be a path from S_Δ to T_Δ in G_Δ . According to the definition of G_Δ , (Δ_1) , \dots , (Δ_4) , we have:

- $x_1 \prec x_2 \prec \dots \prec x_q$,
- $\phi_G(x_i, \dots, x_{i+k-1})$ hold for every $1 \leq i \leq q - k + 1$,
- both $\phi_G^{\min}(x_1, \dots, x_{k-1})$ and $\phi_G^{\max}(x_{q-k+2}, \dots, x_q)$ hold.

Therefore, according to Def. 21, $\{x_1, \dots, x_q\} \in \mathcal{P}_G$.

It is now possible to state the algorithm that enumerates all elements of \mathcal{P}_G :

Algorithm 11: An algorithm for $\text{ENUM_}\mathcal{P}$ for linearly ordered graphs

Input: a linearly ordered graph $G = (V, E)$

Output: an enumeration of \mathcal{P}_G

- 1 Construct $G_\Delta = (V_\Delta, E_\Delta)$, S_Δ and T_Δ
- 2 Construct \mathcal{C} the set of all $X \in \mathcal{P}_G$ of size $< k$
- 3 Let $G'_\Delta = (V'_\Delta, E_\Delta)$ where $V'_\Delta = V_\Delta \cup \mathcal{C}$, $S'_\Delta = S_\Delta \cup \mathcal{C}$ and $T'_\Delta = T_\Delta \cup \mathcal{C}$
- 4 $\text{ENUM_PATHSDAG}(G'_\Delta, S'_\Delta, T'_\Delta)$

Let us examine the complexity of this enumeration algorithm. Given an input graph G , the complexity of $\text{ENUM_}\mathcal{P}$ depends on Theorem 20 and on the pre-processing step, i.e., the construction of the associated directed acyclic graph G_Δ and the construction of the set \mathcal{C} of subsets of vertices $X \in \mathcal{P}_G$ with $|X| < k$.

As explained above, constructing G_Δ consists in constructing V_Δ , E_Δ , S_Δ and T_Δ . There are n^{k-1} tuples of size $k-1$ of vertices of G and so building V_Δ takes a time in $O(n^{k-1})$. The sets S_Δ and T_Δ are computed depending on the complexity of ϕ_G^{\min} and ϕ_G^{\max} , which are by assumption decidable in polynomial time. There are n^k possible edges in G_Δ , building E_Δ depends on n^k and on the complexity of ϕ_G , which is also decidable in polynomial time. Thus, the total construction of G_Δ takes polynomial time since $k = O(1)$.

One can construct the set \mathcal{C} by an exhaustive examination of all the sets of size $< k$, deciding whether such a set is selected can be done in polynomial time since \mathcal{P} is a polynomial vertex set property.

Once G'_Δ , S'_Δ and T'_Δ are available, as stated in Theorem 20, the enumeration of the paths from S'_Δ to T'_Δ in G'_Δ provides an enumeration of all sets in \mathcal{P}_G with linear delay with respect to the original graph since the size of the paths in G'_Δ is linear in the size of the vertex sets they represent in G .

We conclude that $\text{ENUM_}\mathcal{P}$ can be solved with linear delay after a polynomial-time pre-processing. More precisely, if property \mathcal{P} is locally definable through windows of size k , if constraint ϕ is decidable in time $O(n^c)$ for some $c > 0$ and constraints ϕ_G^{\min} and ϕ_G^{\max} are both decidable in time $O(n^d)$ for some $d > 0$, then the pre-processing step which consists in the construction of $G_\Delta = (V_\Delta, E_\Delta)$ requires time $O(\max(n^{k+c}, n^{(k-1)+d}))$. □

Interestingly, this reduction gives us a compact and structured representation of the solutions space, which can be used to solve efficiently algorithmical task other than enumeration. For example, once the DAG associated to G is constructed, one can find a minimum or a maximum set that satisfies property \mathcal{P} , or count the sets satisfying \mathcal{P} in polynomial time.

$\text{MINIMUM_}\mathcal{P}$	
<i>Input :</i>	a graph $G = (V, E)$
<i>Output :</i>	a minimum subset $X \subseteq V$ such that $X \in \mathcal{P}_G$

$\text{MAXIMUM_}\mathcal{P}$	
<i>Input :</i>	a graph $G = (V, E)$
<i>Output :</i>	a maximum subset $X \subseteq V$ such that $X \in \mathcal{P}_G$

$\#\mathcal{P}$	
<i>Input :</i>	a graph G
<i>Output :</i>	$ \mathcal{P}_G $

Finding a minimum (resp. maximum) set satisfying \mathcal{P} can be done by applying a shortest path algorithm (resp. a longest path algorithm) from S to T (for example Disjkstra's algorithm). Counting the sets satisfying \mathcal{P} can be done by applying some counting algorithm, for example Algorithm 12 below.

Algorithm 12: # PATHSDAG

Input: a DAG $G = (V, A)$ with $V = \{x_1, \dots, x_n\}$, a source s and a sink t

Output: the number of paths from s to t

- 1 Let (x_1, \dots, x_n) be a topological ordering of G
- 2 Initialise $Count[x]$ to 0 for every vertex $x \in V$
- 3 $Count[t] = 1$
- 4 **for** $i = n$ to 1 **do**
- 5 **for** each outgoing neighbour y of x_i **do**
- 6 $Count[x_i] = Count[x_i] + Count[y]$
- 7 Output $Count[s]$

Modifying line 4 of Algorithm 11 by the call of Algorithm 12 above or by a shortest or longest path polynomial time algorithm allows us to state the following result.

Corollary 23. *Let (\mathcal{G}, \preceq) be a uniformly ordered set of graphs and \mathcal{P} a vertex set property. If \mathcal{P} is locally definable in polynomial time on (\mathcal{G}, \preceq) , then $\text{MINIMUM}_{\mathcal{P}}$ and $\text{MAXIMUM}_{\mathcal{P}}$ belong to complexity class \mathcal{P} and $\#\mathcal{P}$ to complexity class FP.*

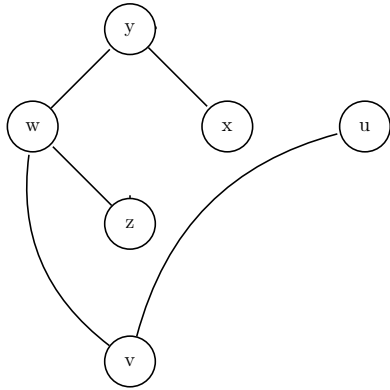
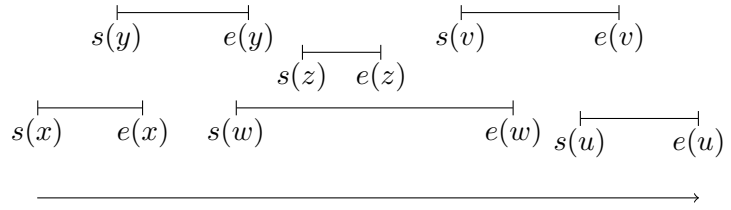
In the following we show how Theorem 22 can be applied on two kinds of intersection graphs, namely interval graphs and permutation graphs.

3.4 MDS, MCDS and MIR in interval graphs

We start by presenting the class of interval graphs and by stating properties that can be defined locally on this class of graphs. We then give a local characterization of minimal dominating sets, minimal connected dominating sets and maximal irredundant sets in interval graphs.

3.4.1 Interval graphs and locally definable basic properties

An *intersection graph* is a graph whose vertices are map onto sets in such a way that two vertices are adjacent if and only if their corresponding sets intersect. The collection of sets related to the vertices is called the *intersection model* of the graph. An intersection graph is an *interval graph* if its intersection model is composed of intervals on the real line. If $G = (V, E)$ is such a graph, we denote by $[s(x), e(x)]$ the interval associated with a vertex x and we assume without loss of generality that all endpoints are pairwise distinct. There is a natural linear order \preceq associated to the vertices of an interval graph: $x \preceq y$ if $s(x) \leq s(y)$. In the following, when dealing with interval graphs we always consider its intersection model and we always implicitly refer to this order.


 Figure 3.13: Interval graph G

 Figure 3.14: Intersection model of G

We say that two intervals are *nested* if one is included in the other. For example, consider the interval graph given in Figure 3.13 and its intersection model given in Figure 3.14. The graph contains a nest: z is included in w . A vertex set X in an interval graph has no *nested intervals* if for all distinct vertices x, y in X , $e(x) < e(y)$ whenever $x \prec y$. Vertex sets without nested intervals have the interesting property that different basic properties can be read locally on them. Moreover, the fact that a set of vertices of an interval graph does not contain any nested intervals is also locally checkable, as stated in the following lemma.

Lemma 24. *Let $G = (V, E)$ be an ordered interval graph, and $X \subseteq V$. The set X has no nested intervals if and only if for all $(x, y) \in \mathbb{W}^2(X)$, $e(x) < e(y)$.*

Proof. One direction is trivial. Conversely, suppose that for all $(x, y) \in \mathbb{W}^2(X)$, $e(x) < e(y)$. Suppose $X = \{x_1 \prec x_2 \prec \dots \prec x_q\}$. Consider $x_i \prec x_j$ with $i > j$. Then, we have $x_i \prec x_{i+1} \prec \dots \prec x_j$, and thus by assumption $e(x_i) < e(x_{i+1}) < \dots < e(x_j)$. \square

Lemma 25. *Let $G = (V, E)$ be an ordered interval graph, and let X be a set of vertices with no nested intervals.*

(Connectivity) X is connected if and only if for any $(x, y) \in \mathbb{W}^2(X)$, $y \in N(x)$.

(Domination) Let v be a vertex in $V \setminus X$. Then, $v \in N(X)$ if and only if:

- if $x \prec v \prec y$ with $(x, y) \in \mathbb{W}^2(X)$, $v \in N(x, y)$;
- if $v \prec \text{Min}(X)$, $v \in N(\text{Min}(X))$;
- if $v \succ \text{Max}(X)$, $v \in N(\text{Max}(X))$.

(Private neighbourhood) Let y be a vertex in X , then

- if $(x, y, z) \in \mathbb{W}^3(X)$, $P_X(y) = P_{\{x,y,z\}}(y)$;
- if $y = \text{Min}(X)$ and $(y, z) \in \mathbb{W}^2(X)$, $P_X(y) = P_{\{y,z\}}(y)$;
- if $y = \text{Max}(X)$ and $(x, y) \in \mathbb{W}^2(X)$, $P_X(y) = P_{\{x,y\}}(y)$.

Proof. (Connectivity) Let (x, y) be in $\mathbb{W}^2(X)$. Because $y \in N(x)$, any two successive vertices in X are adjacent and hence X is connected. Conversely, suppose that $y \notin N(x)$, that is $e(x) < s(y)$. On the one hand, any w in X such that $w \prec x$, verifies $e(w) < e(x)$ since X contains no nested intervals. On the

other hand, any z in X such that $y \prec z$ starts after $s(y)$. This situation prohibits the existence of a path from x to y , thus contradicting the connectivity of X .

(Domination) Let v be a vertex in $V \setminus X$ such that $x \prec v \prec y$ with $(x, y) \in \mathbb{W}^2(X)$. If $v \in N(x, y)$, then clearly $v \in N(X)$. Conversely, suppose that $v \notin N(x, y)$. Then, $e(x) < s(v)$ and $e(v) < s(y)$. For any w in X such that $w \prec x$ we have $e(w) < e(x)$ by assumption on X , thus $e(w) < s(v)$ and hence $v \notin N(w)$. For any z in X such that $y \prec z$ we have $s(y) < s(z)$, thus $e(v) < s(z)$ and hence $v \notin N(z)$. Therefore, $v \notin N(X)$.

Extremal cases, namely when $v \prec \text{Min}(X)$ or $v \succ \text{Max}(X)$, can be handled in a similar way.

(Private neighbourhood) Suppose $(x, y, z) \in \mathbb{W}^3(X)$. It is clear that $P_X(y) \subseteq P_{\{x, y, z\}}(y)$, so let us prove the converse inclusion. Let $u \in P_{\{x, y, z\}}(y)$. Since u is a neighbour of y , but not a neighbour of x , it starts after x ends, $e(x) < s(u)$. Since u is a neighbour of y , but not a neighbour of z , it ends after z starts, $e(u) < s(z)$. Hence, $e(x) < s(u) < e(u) < s(z)$. For any w in X such that $w \prec x$ we have $e(w) < e(x)$ by assumption on X , thus $e(w) < s(u)$ and hence $u \notin N(w)$. For any t in X such that $z \prec t$ we have $s(z) < s(t)$, thus $e(u) < s(t)$ and hence $u \notin N(t)$. Therefore, $u \in N[y] \setminus N[X \setminus \{y\}]$, i.e., $u \in P_X(y)$.

Extremal cases can be handled in a similar way. □

Lemma 26. *Let X be an irredundant set or a minimal connected dominating set of an interval graph. Then X has no nested intervals.*

Proof. Suppose that there exist x, y in X such that $x \preceq y$ and $e(x) > e(y)$. Then $[s(y), e(y)] \subset [s(x), e(x)]$, which is a contradiction with the fact that X is an irredundant set (resp. X a minimal connected dominating set). □

In the remaining of the section, we illustrate our previous method in considering some particular vertex set properties that are locally definable on interval graphs. We start by recalling the known result on minimal dominating sets, adapting it to our frame, and we then turn to new properties.

3.4.2 Interval graphs and minimal dominating sets

We denote by MDS the vertex set property that associates to each graph the collection of its minimal dominating sets. We recall the result obtained by Kante et al. and put forward the local characterization of this property on interval graphs.

Proposition 27 ([66]). *For the class of interval graphs, ENUM_MDS is enumerable with linear delay after a pre-processing in $O(n^3)$.*

Proof. This result was already proved by Kanté et al. [66, Prop. 5], we reformulate it by making explicit the fact that the minimal dominating set property is locally definable in polynomial time on interval graphs. Thus the above proposition follows from Theorem 22.

The key point is that the notion of domination as well as the one of irredundancy become local when we restrict ourselves to properties on interval graphs that do not contain nested intervals.

Let us recall that X is a minimal dominating set of a graph G if:

- every vertex $v \in V \setminus X$ is dominated by X ,
- every vertex $y \in X$ is irredundant, i.e., $P_X(y) \neq \emptyset$.

A minimal dominating set being irredundant, by Lemma 26 we know that it does not contain any nested intervals and this can be verified locally by Lemma 24. Under this "no nested intervals" condition, according to Lemma 25 the notion of domination as well as the one of irredundancy become local. Indeed, domination can be checked through windows of size 2 while irredundancy needs windows of size 3. To

check simultaneously these two properties, we will use windows of size 3. Such a window (x, y, z) will be used in the general case to check that x and y are not nested, that y has a private neighbour and that every vertex of G in the interval $[x, y]$ is dominated. This introduces a slight dissymetry for the extremal windows.

On the one hand the border window (x, y) such that $x = \text{Min}(X)$ will additionally be used to prove that x is irredundant and that all vertices $v \prec x$ are dominated. On the other hand, the border window (y, z) such that $z = \text{Max}(X)$ will additionally be used to prove that z is irredundant, that all vertices $v \succ z$ are dominated but also that z and y are not nested intervals and that all vertices $y \prec v \prec z$ are dominated.

Therefore, according to Lemma 26, Lemma 24 and Lemma 25, minimal dominating sets can be characterized through scanning windows of size 3 on ordered interval graphs. More precisely, given an interval graph $G = (V, E, \preceq)$ and $X \subseteq V$, $X \in \text{MDS}(G)$ if and only if

- for every $(x, y, z) \in \mathbb{W}^3(X)$,
 - (no nested intervals) $e(x) < e(y)$
 - (irredundancy) $P_{\{x,y,z\}}(y) \neq \emptyset$
 - (domination) $\forall a \in]x, y[: a \in N(x, y)$
- for $(x, y) \in \mathbb{W}^2(X)$ such that $x = \text{Min}(X)$,
 - (irredundancy) $P_{\{x,y\}}(x) \neq \emptyset$
 - (domination) $\forall a, a \prec x : a \in N(x)$
- for $(y, z) \in \mathbb{W}^2(X)$ such that $z = \text{Max}(X)$,
 - (no nested intervals) $e(y) < e(z)$
 - (irredundancy) $P_{\{y,z\}}(z) \neq \emptyset$
 - (domination) $\forall a \in]y, z[: a \in N(y, z)$ and $\forall a, a \succ z : a \in N(z)$

Remark 1. Notice that the condition that for every $(x, y) \in \mathbb{W}^2(X)$, $e(x) < e(y)$ (which reflects that there is no nested intervals and which is required to get the local definability of basic properties) is redundant since it follows from the fact that for every $(x, y, z) \in \mathbb{W}^3(X)$, $P_{\{x,y,z\}}(y) \neq \emptyset$ (irredundancy condition), therefore we can omit it. This condition will also be redundant in the following characterizations, and thus we will omit it.

We can decide in polynomial time if a subset of vertices is a minimal dominating set or not, so MDS is a polynomial property. The characterization above shows that MDS is locally definable on the class of interval graphs. Observe that this characterization coincides exactly with the one provided in the paper by Kanté *et al.* [66]. Like in [66], let us show that it is locally definable in time $O(n^3)$. We analyse the time to compute all pairs and triples satisfying the conditions of irredundancy and domination above. We know that there are $O(n^3)$ such tuples. Observe that the verification of the constraint of irredundancy and the constraint of domination can both be performed in linear time. To check that $P_{\{x,y,z\}}(y) \neq \emptyset$ we can examine all neighbours of y and compare their extremities with the ones of x and z . For similar reasons given a , checking that $a \in N(x, y)$ takes a constant time. Since this has to be checked for all a in between x and y , domination can be checked in linear time. This gives a total time $O(n^4)$. Actually, appropriate pre-computations allow to reduce the constraints' verification to constant time.

For all vertex x in G one can compute and store in a table \mathbb{T} : $N(x)$, $x^+ = \{u \mid s(u) > e(x)\}$ and $x^- = \{u \mid e(u) < s(x)\}$ in $O(n^2)$. During the calculation of $N(x)$, we memorise a^x such that $e(a^x) = \min\{e(w) \mid w \in N(x)\}$ and b^x such that $s(b^x) = \max\{s(w) \mid w \in N(x)\}$. Once x^+ and x^- are

known, because the vertices of G are ordered, for all (x, y) in G the set $x^+ \cap y^-$ can be computed and stored in $O(n^3)$. During this calculation we memorise $c^{x,y}$ such that $s(c^{x,y}) = \min\{s(w) \mid w \in x^+ \cap y^-\}$ and $d^{x,y}$ such that $e(d^{x,y}) = \max\{e(w) \mid w \in x^+ \cap y^-\}$.

With this table \mathbb{T} at hand, checking whether an ordered pair $(x \prec y)$ or an ordered triple $(x \prec y \prec z)$ of vertices satisfies the conditions of domination and irredundancy above takes a constant time. In fact, for domination, checking if all vertices between x and y are dominated, boils down to verify that $x^+ \cap y^- = \emptyset$. Testing if the domination is respected on the right of $Min(X)$ boils down to verify that $Min(X)^- = \emptyset$ and testing if the domination is respected on the left of $Max(X)$ boils down to verify that $Max(X)^+ = \emptyset$. Sets $x^+ \cap y^-$, $Min(X)^-$ and $Max(X)^+$ have all been calculated and stored in \mathbb{T} during the step of pre-calculation, thus verifying if they are empty takes a constant time. For irredundancy, observe that checking if $P_{\{x,y,z\}}(y) \neq \emptyset$, boils down to verify that $x^+ \cap N[y] \cap z^- \neq \emptyset$. Testing if $x^+ \cap N[y] \cap z^- \neq \emptyset$ can be done by verifying the following: if $y \in N(x)$ (i.e., $s(y) < e(x)$), then $c^{x,z} \in N(y)$ (i.e., $s(c^{x,z}) < e(y)$) and if $y \in N(z)$ (i.e., $s(z) < e(y) < e(z)$), then $d^{x,z} \in N(y)$ (i.e., $e(d^{x,z}) > s(y)$). Checking if $P_{\{x,y\}}(y) \neq \emptyset$ with $x = Min(X)$, boils down to verify that $N(x) \cap y^- \neq \emptyset$. Testing if $N(x) \cap y^-$ is non empty can be done by verifying that $y \notin N(a^x)$, which can be checked with $s(y) > e(a^x)$. Finally, checking if $P_{\{y,z\}}(y) \neq \emptyset$ with $z = Max(X)$, boils down to verify that $N(z) \cap y^+ \neq \emptyset$. Testing if $N(z) \cap y^+$ is non empty can be done by verifying that $y \notin N(b^z)$, which can be checked with $e(y) < s(b^z)$.

Therefore ordered pairs and triples satisfying the conditions of irredundancy and domination above can be all computed in time $O(n^3)$. Hence, property MDS is locally definable in time $O(n^3)$ on the class of interval graphs. Moreover, there are $O(n^2)$ sets of one or two vertices that are candidates to be MDS. Verifying whether each of them is a minimal dominating sets can be done in linear time. So, dealing with minimal dominating sets of size less than 3 requires time $O(n^3)$. We conclude that $ENUM_MDS$ is enumerable with linear delay after a pre-processing in $O(n^3)$. □

3.4.3 Interval graphs and minimal connected dominating sets

We denote by MCDS the vertex set property that associates to each graph the collection of its minimal connected dominating sets. Non surprisingly, since connectivity is also locally definable on interval graphs, we get a local characterization of minimal connected dominating sets on such graphs.

Proposition 28. *For the class of interval graphs, $ENUM_MCDS$ is enumerable with linear delay after a pre-processing in $O(n^3)$.*

Proof. We prove that the minimal connected dominating set property is locally definable in polynomial time on interval graphs. The above proposition then follows from Theorem 22.

Let us consider interval graphs equipped with their usual linear order defined above. Given such a graph $G = (V, E, \preceq)$ and $X \subseteq V$, $X \in MCDS(G)$ if

- X is connected
- every vertex $v \in V \setminus X$ is dominated by X
- for all $y \in X$, either $P_X(y) \neq \emptyset$ or $X \setminus \{y\}$ is disconnected.

According to Lemma 26, Lemma 24 and Lemma 25, the notions of connectivity, domination and private neighbourhood are all locally definable in interval graphs through windows of size at most 3. Given a connected set X , first observe that if X contains no nested intervals, X cannot be disconnected by the removal of any of its two extremal elements. Now given $y \in X$ with $(x, y, z) \in \mathbb{W}^3(X)$, the set

$X \setminus y$ is no longer connected if and only if $z \notin N(x)$. We conclude that we will need windows of size 3 in order to characterize minimal connected dominating sets.

A general window (x, y, z) will be used to check the connectivity between x and y . This introduces a slight dissymetry, thus the extremal window $(y, Max(X))$ will be used to check the connectivity between y and $Max(X)$.

Observe that if X is connected, then all vertices v such that $Min(X) \prec v \prec Max(X)$ are dominated. Hence, if X is connected, in order to check if X is a dominating set, we need only to verify that each $v \prec Min(X)$ and each $v \succ Max(X)$ are dominated. This will be done through extremal windows $(Min(X), y)$ and $(y, Max(X))$.

Now, it remains to deal with minimality. By the removal of one element either we loose the connectivity (when an internal element is removed) or we keep the connectivity and loose the domination (when an extremal element is removed). Thus, a window (x, y, z) will also be used to check that $X \setminus y$ is not connected anymore, in checking that $z \notin N(x)$. The extremal windows $(Min(X), y)$ and $(y, Max(X))$ will be used to check that $X \setminus Min(X)$ and $X \setminus Max(X)$ are not dominating anymore, i.e., $P_X(Min(X)) \neq \emptyset$ and $P_X(Max(X)) \neq \emptyset$.

From those observations and by Lemma 26, Lemma 24 and Lemma 25, we get that $X \in MCDS(G)$ if and only if

- for every $(x, y, z) \in \mathbb{W}^3(X)$ we have
 - (connectivity) $y \in N(x)$;
 - (minimality) $z \notin N(x)$
- for $(x, y) \in \mathbb{W}^2(X)$ such that $x = Min(X)$ we have
 - (domination) $\forall a \prec x, a \in N(x)$;
 - (minimality) $P_{\{x,y\}}(x) \neq \emptyset$;
- for $(y, z) \in \mathbb{W}^2(X)$ such that $z = Max(X)$ we have
 - (connectivity) $z \in N(y)$;
 - (domination) $\forall a \succ z, a \in N(z)$;
 - (minimality) $P_{\{y,z\}}(z) \neq \emptyset$.

Notice that the fact that for all $(x, y) \in \mathbb{W}^2(X)$, $e(x) < e(y)$ (which reflects that there is no nested intervals and which is required to get the local definability of basic properties) is omitted. Indeed it is easy to see that it follows from the fact that for every $(x, y, z) \in \mathbb{W}^3(X)$, $y \in N(x)$ and $z \notin N(x)$ (connectivity and minimality conditions).

The complexity analysis of the pre-processing step for the enumeration of MCDS is analogous to, and even simpler than the one made in the case of the enumeration of MDS. We can decide in polynomial time if a subset of vertices is a minimal connected dominating set or not, so MCDS is a polynomial property. The characterization above shows that MCDS is locally definable on the class of interval graphs. There are $O(n^3)$ pairs and triples satisfying the conditions of minimal connected domination above. The elementary instructions to be carried out while checking these conditions are either *test of belonging* to given neighbourhood (e.g. Does $z \in N(x)$?) or *vacuity test* on private neighbourhoods (e.g. Is $P_{\{x,y\}}(y) = \emptyset$?). The former can be done in constant time, since one can check whether $z \in N(x)$ by comparing the relative positions of $s(x), s(z), e(x), e(y)$. Furthermore, we have seen in the complexity analysis of MDS

that irredundancy can be verified in constant time after a pre-processing (the calculation of \mathbb{T}) in $O(n^3)$. Therefore ordered pairs and triples satisfying the conditions of minimal connected domination above can be all computed in time $O(n^3)$. Hence, property MCDS is locally definable in time $O(n^3)$ on the class of interval graphs. Moreover, there are $O(n^2)$ sets of one or two vertices that are candidates to be MCDS. Verifying whether each of them is a minimal connected dominating sets can be done in linear time. So, dealing with minimal connected dominating sets of size less than 3 requires time $O(n^3)$ too. We conclude that ENUM_MCDS is enumerable with linear delay after a pre-processing in $O(n^3)$. \square

3.4.4 Interval graphs and maximal irredundant sets

We denote by MIR the vertex set property that associates to each graph the collection of its maximal irredundant sets. We prove that maximal irredundant sets are efficiently enumerable for interval graphs.

Proposition 29. *For the class of interval graphs, ENUM_MIR is enumerable with linear delay after a pre-processing in $O(n^5)$.*

Proof. We prove that MIR is polynomial time locally definable on the class of interval graphs. Thus the above proposition follows from Theorem 22.

Let $G = (V, E)$ be an ordered interval graph and $X \subseteq V$. The set X is a maximal irredundant set if:

- (irredundancy) for all $y \in X$, $P_X(y) \neq \emptyset$,
- (maximality) for all $a \in V \setminus X$ either $P_{X \cup a}(a) = \emptyset$ or $P_{X \cup a}(s) = \emptyset$ for some $s \in X$.

The definition above makes use of private neighbourhood of a vertex w.r.t. a set. Let us remind that the vacuity of such a private neighbourhood can be checked locally as far as the set contains no nested intervals. Therefore, according to Lemma 26 we can consider the following equivalent definition.

The set X is a maximal irredundant set if and only if:

- X has no nested intervals,
- (irredundancy) for all $y \in X$, $P_X(y) \neq \emptyset$,
- (maximality) for all $a \in V \setminus X$,

either $X \cup a$ contains nested intervals,

or $X \cup a$ has no nested intervals and

either $P_{X \cup a}(a) = \emptyset$

or $P_{X \cup a}(s) = \emptyset$ for some $s \in X$.

One can verify locally that X contains no nested intervals by Lemma 24. Under this condition one has a local characterization of private neighbourhood, see Lemma 25. Moreover, when X contains no nested intervals, $X \cup a$ contains nested intervals if, supposing $x \prec a \prec y$ with $(x, y) \in \mathbb{W}^2(X)$, we have either $e(x) > e(a)$ or $e(a) > e(y)$. It remains to check that an s in X such that $P_X(s) \neq \emptyset$ and $P_{X \cup a}(s) = \emptyset$ can be found locally. Actually such an s cannot be found anywhere else but in a 2-window containing a . More precisely when $P_X(s) \neq \emptyset$ for all $s \in X$, supposing $x \prec a \prec y$ with $(x, y) \in \mathbb{W}^2(X)$, $P_{X \cup a}(s) = \emptyset$ can only occur when $s = x$ or $s = y$. Therefore the last condition of the above characterization can be written $P_{X \cup a}(a) = \emptyset$ or $P_{X \cup a}(x) = \emptyset$ or $P_{X \cup a}(y) = \emptyset$. Deciding the vacuity of a private neighbourhood requires window of size 3. Therefore a window a size 4 is sufficient to check maximality.

As a consequence, a maximal irredundant set can be entirely defined locally on interval graphs. Irredundancy is locally checked through windows of size 3 and maximality through windows of size 4. The property of being maximal irredundant will thus be checked through windows of size 4. On window (w, x, y, z) the irredundancy will be checked on x and the maximality will be checked between x and y . On border window (w, x, y, z) where $w = \text{Min}(X)$ we will ensure that w is irredundant and that the maximality is respected between w, x and on the left of w . Finally on border window (w, x, y, z) where $z = \text{Max}(X)$ we will ensure that not only y but also z are both irredundant and that the maximality is respected between y, z and on the right of z .

From these observations, from Lemma 26, Lemma 24 and Lemma 25, we get that $X \in \text{MIR}(G)$ if and only if:

- For every $(w, x, y, z) \in \mathbb{W}^4(X)$,
 - (irredundancy) $P_{\{w,x,y\}}(x) \neq \emptyset$,
 - (maximality) for all a such that $x \prec a \prec y$,
 - either: $e(x) > e(a)$ or $e(a) > e(y)$
 - or: $P_{\{x,a,y\}}(a) = \emptyset$ or $P_{\{w,x,a\}}(x) = \emptyset$ or $P_{\{a,y,z\}}(y) = \emptyset$
- For every $(x, y, z) \in \mathbb{W}^3(X)$ such that $x = \text{Min}(X)$,
 - (irredundancy) $P_{\{x,y\}}(x) \neq \emptyset$,
 - (maximality)
 - * for all a such that $a \prec x$,
 - either: $e(a) > e(x)$
 - or: $P_{\{a,x\}}(a) = \emptyset$ or $P_{\{a,x,y\}}(x) = \emptyset$
 - * for all a such that $x \prec a \prec y$,
 - either: $e(x) > e(a)$ or $e(a) > e(y)$
 - or: $P_{\{x,a,y\}}(a) = \emptyset$ or $P_{\{x,a\}}(x) = \emptyset$ or $P_{\{a,y,z\}}(y) = \emptyset$
- For every $(x, y, z) \in \mathbb{W}^3(X)$ such that $z = \text{Max}(X)$,
 - (irredundancy) $P_{\{x,y,z\}}(y) \neq \emptyset$ and $P_{\{y,z\}}(z) \neq \emptyset$
 - (maximality)
 - * for all a such that $a \succ z$,
 - either: $e(z) > e(a)$
 - or: $P_{\{z,a\}}(a) = \emptyset$ or $P_{\{y,z,a\}}(z) = \emptyset$
 - * for all a such that $y \prec a \prec z$,
 - either: $e(y) > e(a)$ or $e(a) > e(z)$
 - or: $P_{\{y,a,z\}}(a) = \emptyset$ or $P_{\{x,y,a\}}(y) = \emptyset$ or $P_{\{a,z\}}(z) = \emptyset$

As for minimal dominating sets, the condition that for every $(x, y) \in \mathbb{W}^2(X)$, $e(x) < e(y)$ (which reflects that there is no nested intervals and which is required to get the local definability of basic properties) follows from the fact that for every $(w, x, y, z) \in \mathbb{W}^4(X)$, $P_{\{w,x,y\}}(x) \neq \emptyset$ (irredundancy condition) and thus we omitted it.

We can decide in polynomial time if a subset of vertices is a maximal irredundant set or not, so MIR is a polynomial property. The characterization above shows that MIR is locally definable on the class of interval graphs. Let us show that it is locally definable in time $O(n^5)$. There are $O(n^4)$ pairs and triples satisfying the conditions of irredundancy and maximality above. Those conditions are expressed through constraints of irredundancy, redundancy and of the form $e(x) > e(a)$. As we observed in the complexity analysis of MDS through the calculation of \mathbb{T} , each of these constraints takes a constant time after a pre-processing in $O(n^3)$. For maximality such constraints need to be verified for all n possible $a \in V \setminus X$. Therefore ordered pairs and triples satisfying the conditions of maximal irredundancy above can be all computed in time $O(n^5)$. Hence, property MIR is locally definable in time $O(n^5)$ on the class of interval graphs. Moreover, there are $O(n^3)$ sets of one, two or three vertices that are candidates to be MIR. Verifying whether each of them is a maximal irredundant sets can be done in linear time. So, dealing with maximal irredundant sets of size less than 4 requires time $O(n^4)$. We conclude that ENUM_MIR is enumerable with linear delay after a pre-processing in $O(n^5)$. \square

Let us conclude this section by noticing that our technique provides a polynomial time algorithm for finding a *minimum* (that is, of minimal cardinality) connected dominating set, as well as a *maximum* irredundant set, on an interval graph. It also provides an algorithm for counting the minimal connected dominating sets and the maximal irredundant sets in polynomial time on such a graph.

Corollary 30. *On interval graphs the problem of finding a minimum connected dominating set, as well as the problem of finding a maximum irredundant set, can be solved in polynomial time. Moreover, it is possible to count the minimal connected dominating sets and the maximal irredundant sets in polynomial time on such graphs.*

Proof. Let us deal with the minimum connected dominating set problem (the proofs for the maximum irredundant set problem and the counting problems are analogous). Given an interval graph, we use the algorithm invoked in the proof of Proposition 28 (and detailed in Theorem 22), which enumerates paths in a DAG. In this algorithm we replace the call to ENUM_PATHSDAG by a call to a polynomial-time procedure computing a *shortest* path. This provides a minimum connected dominating set. \square

3.5 MDS, MCDS and MIR in permutation graphs

We start by presenting the class of permutation graphs and by stating properties that can be defined locally on this class of graphs. We then give a characterization of minimal dominating sets, minimal connected dominating sets and maximal irredundant sets in permutation graphs.

3.5.1 Permutation graphs and locally definable basic properties

An intersection graph is a *permutation graph* if its intersection model is composed of straight lines between two parallels. If $G = (V, E)$ is such a graph and $x \in V$, we denote by $L_G(x)$ the segment in the intersection model of G corresponding to x . We number the endpoints of the segments from left to right and we denote by $b(x)$ and by $t(x)$ the endpoints of the intersection model on the bottom line and top line respectively. All endpoints are assumed to be different without loss of generality. We order the vertices of G by their bottom line endpoints: $x \preceq y$ if $b(x) \leq b(y)$.

An example of permutation graph and its intersection model is given in Figures 3.15 and 3.16.

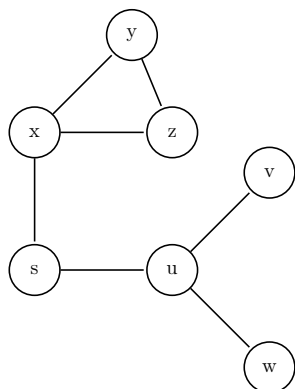


Figure 3.15: Permutation graph G

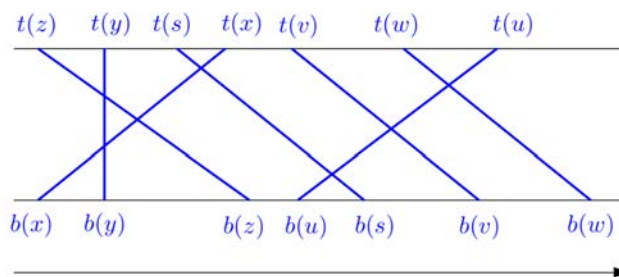


Figure 3.16: Intersection model of G

As for interval graphs, we are interested in expressing some properties on permutation graphs locally through sliding windows of fixed size. Permutation graphs are combinatorially more involved than interval graphs and we will need windows of larger size.

In the case of interval graphs a special care has to be given to the border windows depending on the minimal number of vertices needed to check the property considered and on the number of vertices available. For example, if $X = \{x_1 \prec \dots \prec x_q\} \subseteq V(G)$, expressing the irredundancy of vertex x_i requires at least three vertices: x_i , its predecessor and its successor, and so, the general window is of size 3. But for the first vertex x_1 (resp. the last vertex x_q), only x_2 (resp. x_{q-1}) is available, there does not exist a predecessor (resp. a successor) and so left border window (resp. right border window) is of size 2. In the case of permutation graphs, we will need windows of much larger sizes and so the number of extremal cases will increase. For example, for expressing the irredundancy of some vertex x_i , we will need at least 7 vertices: x_i , its 3 predecessors and its 3 successors, and so the general window will be of size 7. As for interval graphs, for vertex x_1 no predecessor is available. But this is not the only extremal case to consider: for vertex x_3 only two predecessors are available and for x_2 only one.

So, before going further let us introduce the notion of surrounding-window that will make easier the discussion on the extremal cases.

Given a set of vertices X from a permutation graph, we define the X -distance between u and v , $dist_X(u, v)$, by:

$$dist_X(u, v) = \begin{cases} 0 & \text{if } u = v; \\ 1 + \text{card}(\]u, v[\cap X) & \text{otherwise;} \end{cases}$$

where $\]u, v[$ denotes the set of vertices lying in between u and v , whatever u is greater or smaller than v .

Definition 31. Let (G, \preceq) be an ordered graph, $X \subseteq V(G)$ and $v \in V(G)$. A k -surrounding window of X for v , denoted by $sw_X^k(v)$, is the window composed of all vertices at X -distance at most k of v . Vertex v belongs to the surrounding window only if $v \in X$.

The idea is that a k -surrounding window of X for v is a window that frames v . The surrounding window is composed of two windows of size at most k : one with the k predecessors of v in X and one with the k successors of v in X . Each of these windows can be of size smaller than k when v has less than k

predecessors or successors. Observe that if $v \in V \setminus X$, the surrounding window is of size at most $2k$ and if $v \in V$ it is of size at most $2k + 1$.

Example. If $G = (V, E)$ is an ordered graph and if $X = \{x_1 \prec x_2 \prec \dots \prec x_{12}\}$, then the tuple $(x_2, x_3, x_4, x_5, x_6)$ is the 2-surrounding window of vertex $x_4 \in X$, the tuple (x_5, x_6, x_7, x_8) is the 2-surrounding window of vertex v such that $x_6 \prec v \prec x_7$. Truncated surrounding windows can also occur, when there are not enough predecessors or successors available. For instance, the tuple $(x_9, x_{10}, x_{11}, x_{12})$ is the 2-surrounding window of vertex $x_{11} \in X$, and (x_1, x_2, x_3, x_4) is the 3-surrounding window of vertex $x_1 \prec v \prec x_2$. \triangleleft

Lemma 32. Let G be a permutation graph. For all vertices x, y, z such that $x \prec y \prec z$:

1. if $z \in N(x)$, then $y \in N(x)$ or $y \in N(z)$;
2. if $y \in N(x)$ and $z \in N(y)$, then $z \in N(x)$.

Proof. (1) Because vertices x, y, z are such that $x \prec y \prec z$ and $z \in N(x)$, we have $t(z) \prec t(x)$. Thus, if $t(y) \prec t(z)$, we have $y \in N(x)$, if $t(z) \prec t(y) \prec t(x)$, we have $y \in N(x)$ and $y \in N(z)$ and if $t(y) \succ t(x)$, we have $y \in N(z)$. (2) From $x \prec y \prec z$, $y \in N(x)$ and $z \in N(y)$, we get $t(z) \prec t(y) \prec t(x)$ and therefore, $z \in N(x)$. This proof is illustrated by Figures 3.17 and 3.18 below. \square

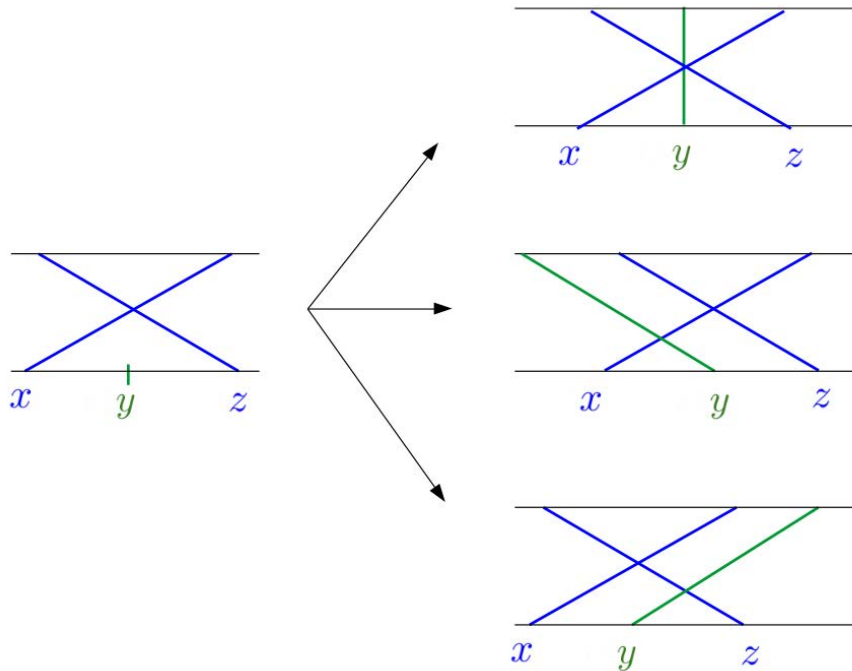


Figure 3.17: Lemma 32 (1)

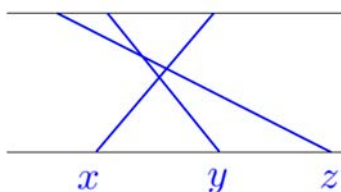


Figure 3.18: Lemma 32 (2)

In the case of interval graphs we have seen that basic properties can be defined locally if they contain no nested intervals and this last property can be checked locally. For permutation graphs, the fact that they do not contain any vertex of degree greater than 2 is the characteristic that enables us to define our basic properties locally. This characteristic property can be verified locally on permutation graphs.

Let us recall that for any set of vertices X and any $x \in X$, we denote by $\deg_X(x)$ the degree of x in X and we set $\deg(X) = \max\{\deg_X(x) \mid x \in X\}$.

Lemma 33. *Let $G = (V, E)$ be an ordered permutation graph, and $X \subseteq V$. Then $\deg(X) \leq 2$ if and only if for all $W \in \mathbb{W}^6(X)$, $\deg(W) \leq 2$.*

Proof. The left-to-right direction is clear, and we only have to prove:

$$\deg(X) \geq 3 \Rightarrow \exists W \in \mathbb{W}^6(X) : \deg(W) \geq 3.$$

A set X of degree ≥ 3 contains windows of degree ≥ 3 (X itself, for instance). Let W be such a window of minimal size and let x and u denote its endpoints, with $x < u$.

We first prove that one of these endpoints has degree ≥ 3 . By assumption W contains a vertex v of degree ≥ 3 . If v is an endpoint of W , we are done; if $x < v < u$, then x is adjacent to v , otherwise v would still have three neighbours in the window $W \setminus \{x\}$, thus contradicting the minimality of W . By symmetry, the same holds for u and consequently both x and u are adjacent to v . Hence they are adjacent to each other, by Lemma 32 (2). Besides, v has a third neighbour in W . Call it y and assume, w.l.o.g., that $x < y < v < u$. Since both y and u are adjacent to v , they are adjacent to each other, still by Lemma 32 (2). Finally, x, y, v belong to $N(u)$ and hence, $\deg_W(u) \geq 3$.

Thus, we can assume w.l.o.g. that W contains four vertices $x < y < v < u$ such that x, u are its extremities and $x, y, v \in N(u)$. Now, suppose W contains three more vertices $a, b, c \notin \{x, y, v, u\}$. None of these vertices can belong to $N(u)$, otherwise u would still have degree ≥ 3 in the window $W \setminus \{x\}$, thus contradicting the minimality of W . Therefore a, b, c are adjacent to x according to Lemma 32 (1). This entails that $W \setminus \{u\}$ contains a vertex of degree 3, thus contradicting the minimality of W . Hence W cannot contain three additional vertices and thus it belongs to $\mathbb{W}^k(X)$ for $k \leq 6$. \square

We are now in a position to show that basic properties can be defined locally on permutation graphs.

Lemma 34. *Let $G = (V, E)$ be an ordered permutation graph, and $X = \{x_1 \prec \dots \prec x_q\} \subseteq V$ such that $\deg(X) \leq 2$.*

(Connectivity) X is connected if and only if

$$- \text{for any } (x_{i-1}, \dots, x_{i+2}) \in \mathbb{W}^4(X), \{x_{i-1}, x_i\} \cap N(x_{i+1}, x_{i+2}) \neq \emptyset$$

- $x_1 \in N(x_2, x_3)$
- $x_q \in N(x_{q-1}, x_{q-2})$

(Domination) Let v be a vertex in $V \setminus X$. Then, $v \in N(X)$ if and only if

$$v \in N(sw_X^3(v))$$

(Private neighbourhood) Let x_i be a vertex in X . Then

$$P_X(x_i) = P_{sw_X^3(x_i)}(x_i)$$

Proof. (Connectivity) Suppose that X is connected and that there exists $x_i \in X$ with $(x_{i-1}, \dots, x_{i+2}) \in \mathbb{W}^4(X)$ and such that $\{x_{i-1}, x_i\} \cap N(x_{i+1}, x_{i+2}) = \emptyset$. We prove that $\{x_1, \dots, x_i\}$ and $\{x_{i+1}, \dots, x_q\}$ are disconnected, thus getting a contradiction. First notice that for all $j > i + 2$, we have $x_j \notin N(x_i)$. Otherwise, by Lemma 32, x_j would intersect x_i , x_{i+1} and x_{i+2} and so $\deg_X(x_j) \geq 3$, which contradicts the assumption on X . Similarly $x_j \notin N(x_{i-1})$, otherwise x_j would intersect x_{i-1} , x_{i+1} and x_{i+2} and so $\deg_X(x_j) \geq 3$ and thus, contradiction. Finally, we have also $x_j \notin N(x_l)$ for $l < i - 1$. Indeed, if $x_j \in N(x_l)$, since we just showed that $x_j \notin N(x_{i-1}, x_i)$, then by Lemma 32 $x_l \in N(x_{i-1}) \cap N(x_i)$ and hence $\deg_X(x_l) \geq 3$, contradiction. Until now we proved that $\{x_1, \dots, x_i\}$ and $\{x_{i+3}, \dots, x_q\}$ are disconnected. It is easy to see that $\{x_{i+1}, x_{i+2}\}$ does not intersect $\{x_1, \dots, x_i\}$ either, thus proving that $\{x_1, \dots, x_i\}$ and $\{x_{i+1}, \dots, x_q\}$ are disconnected: by hypothesis $\{x_{i+1}, x_{i+2}\}$ do not intersect $\{x_{i-1}, x_i\}$. Moreover, $\{x_{i+1}, x_{i+2}\}$ do not intersect x_l with $l < i - 2$ either. Otherwise x_l would intersect x_{i-1} , x_i and $\{x_{i+1}, x_{i+2}\}$, thus $\deg_X(x_l) \geq 3$, contradiction. We conclude that $\{x_1, \dots, x_i\}$ and $\{x_{i+1}, \dots, x_q\}$ are disconnected.

Now, suppose that we have $x_1 \in N(x_2, x_3)$ and $x_q \in N(x_{q-1}, x_{q-2})$ and for all $(x_{i-1}, \dots, x_{i+2}) \in \mathbb{W}^4(X)$, $\{x_{i-1}, x_i\} \cap N(x_{i+1}, x_{i+2}) \neq \emptyset$ and let us show that X is connected. We show that for all $x_i \in X$, there is a path that connects x_i to x_{i+1} . If $x_i \in N(x_{i+1})$ then we are done. If $x_i \notin N(x_{i+1})$ and $x_i \in N(x_{i+2})$, then x_{i+2} intersects x_{i+1} and hence (x_i, x_{i+2}, x_{i+1}) is a path connecting x_i to x_{i+1} . Now consider the case where $x_i \notin N(x_{i+1}, x_{i+2})$. By assumption $x_{i-1} \in N(x_{i+1}, x_{i+2})$. If $x_{i-1} \in N(x_{i+1})$, then since $x_i \notin N(x_{i+1})$, x_{i-1} intersects x_i and so the path that connects x_i to x_{i+1} is (x_i, x_{i-1}, x_{i+1}) . If $x_{i-1} \notin N(x_{i+1})$ and $x_{i-1} \in N(x_{i+2})$, since $x_i \notin N(x_{i+2})$ then by Lemma 32, x_i and x_{i-1} intersect. Moreover since $x_{i+1} \notin N(x_{i-1})$, then x_{i+1} intersects x_{i+2} . Thus, the path $(x_i, x_{i-1}, x_{i+2}, x_{i+1})$ connects x_i to x_{i+1} .

(Domination) By contradiction suppose that $v \in N(X)$ and $v \notin N(sw_X^3(v))$. Let us say that $x_i \preceq v \preceq x_{i+1}$. There exists $x \in X$ such that $x \preceq x_{i-2}$ or $x \succeq x_{i+3}$ and $v \in N(x)$. If $x \preceq x_{i-2}$, by Lemma 32, we have that x_{i-1}, x_{i-2}, x_i all intersect x and so $\deg_X(x) \geq 3$, which is our contradiction. The reasoning is the same if $x \succeq x_{i+3}$: x intersects $x_{i+1}, x_{i+2}, x_{i+3}$ and so $\deg_X(x) \geq 3$.

(Private neighbourhood) Because $sw_X^3(x_i)$ is a subset of X , we have $P_X(x_i) \subseteq P_{sw_X^3(x_i)}(x_i)$. Now let us show that for all $v \in P_{sw_X^3(x_i)}(x_i)$, $v \in P_X(x_i)$. Take $v \in P_{sw_X^3(x_i)}(x_i)$. We suppose that $v \prec x_i$ (the case where $v \succ x_i$ can be proven in a similar way). We will show that there does not exist any $x_j \in X$ with $j > i + 3$ or $j < i - 3$ such that $x_j \in N(v)$. The first case is due to Lemma 32: if there exists $x_j \in X$ with $j > i + 3$ such that $x_j \in N(v)$, then because v does not intersect x_{i+1}, x_{i+2} and x_{i+3} , we have $x_{i+1}, x_{i+2}, x_{i+3} \in N(x_j)$ and so $\deg_X(x_j) \geq 3$, which is our contradiction. Now let us show that there does not exist any $x_j \in X$ with $j < i - 3$ such that $x_j \in N(v)$. If $v \preceq x_{i-3}$, by Lemma 32 x_{i-3}, x_{i-2} and x_{i-1} intersect x_i and so $\deg_X(x_i) \geq 3$. If $x_{i-3} \preceq v \preceq x_{i-2}$, first observe that because of Lemma 32, x_{i-2} and x_{i-1} are neighbours of x_i . If there exists $x_j < i - 3$ such that $x_j \in N(v)$, then $x_j \in N(x_i)$ and we have our contradiction because $\deg_X(x_i) \geq 3$. If $x_{i-2} \preceq v \preceq x_{i-1}$, first observe that because of Lemma 32, x_{i-1} is

a neighbour of x_i . If there exists $x_j < i - 3$ such that $x_j \in N(v)$, then x_j intersects x_{i-3} , x_{i-2} and x_i and $\deg_X(x_j) \geq 3$. Finally, if $x_{i-1} \preceq v \preceq x_i$, notice that if there exists $x_j < i - 3$ such that $x_j \in N(v)$, then by Lemma 32, x_{i-3} , x_{i-2} and x_{i-1} all intersect x_j and so $\deg_X(x_j) \geq 3$. \square

Lemma 35. *Let G be a permutation graph and X a subset of its vertices. If X is an irredundant set or a minimal connected dominating set of G , then $\deg(X) \leq 2$.*

Proof. Let X be an irredundant or a minimal connected dominating set and suppose by contradiction that there exists $w \in X$, $\deg_X(w) > 2$. Let us denote by x, y, z three neighbours of w in X and without loss of generality suppose that $x \prec y \prec z \prec w$. There are two cases to consider: 1) no two elements in $\{x, y, z\}$ intersect each other or 2) at least two elements of $\{x, y, z\}$ intersect each other. In case 1) vertex y cannot have any private neighbour. So X is not irredundant. The set $X \setminus y$ is still dominating and it is also still connected. Indeed, a neighbour of y is necessarily a neighbour either of x , z or w and we have that $x, z \in N(w)$. Therefore, each path, which goes through y in X can be derived in $X \setminus y$ using x, z and w . So, X is not a minimal connected dominating set. Now, the arguments are the same in case 2), there are only more configurations to consider. If x and y intersect each other, then y has no private neighbour and $X \setminus y$ is still connected for the same reasons than in case 1). If y and z intersect, then it is z that has no private neighbour. Each path, which goes through z in X can be derived in $X \setminus z$ using x, y and v and so $X \setminus z$ is still connected. The reasoning is the same if x and z intersect. \square

Figure 3.19 below summarizes Lemma 35 above and its proof. We suppose that $x \prec y \prec z \prec w$ belong to X a subset of vertices of G a permutation graph. Vertex w is such that $\deg_X(w) = 3$. In each configuration the vertex in red cannot have a private neighbour with respect to $\{x, y, z, w\} \subseteq X$ and deleting it does not break the connectivity. Hence, subset X can neither be an irredundant set nor a minimal connected dominating set.

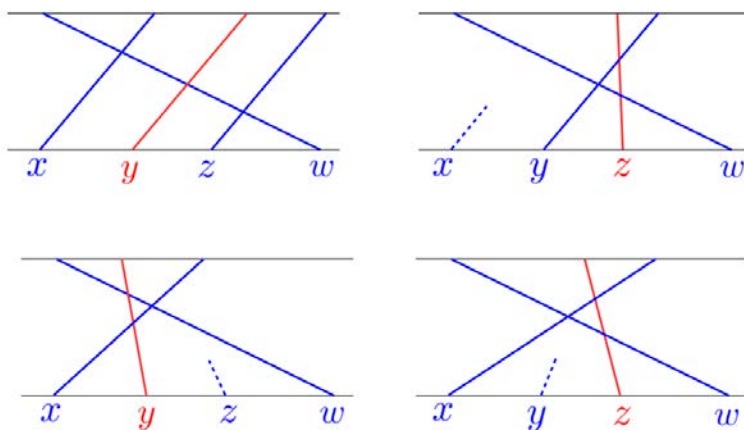


Figure 3.19: Lemma 35, $\deg_X(w) > 2$

In the remaining of the section, we show that minimal connected dominating sets and maximal irredundant sets are locally definable on permutation graphs in polynomial time, and thus thanks to Theorem 22 enumerable with linear delay.

3.5.2 Permutation graphs and minimal dominating sets

Let us recall that MDS states for the vertex set property that associates to each graph the collection of its minimal dominating sets. We recall the result obtained by Kante et al. and put forward the local characterization of this property on permutation graphs.

Proposition 36 ([66]). *For the class of permutation graphs, ENUM_MDS is enumerable with linear delay after a pre-processing in $O(n^8)$.*

Proof. This result was already proved by Kanté et al. [66, Prop. 17], we reformulate it by making explicit the fact that the minimal dominating set property is locally definable in polynomial time on permutation graphs. Thus the above proposition follows from Theorem 22.

By the definition of a minimal dominating set and since every such set in a permutation graph has degree 2 (Lemma 35), $X \in \text{MDS}(G)$ if and only if:

- $\deg(X) \leq 2$
- X is dominating.
- for every $y \in X$, $P_X(y) \neq \emptyset$.

According to Lemma 33 the condition on the degree can be checked through a window of size 6. Under this bounded degree condition the notion of domination as well as the one of irredundancy become local as observed in Lemma 34. One can see that domination can be checked through windows of size 6, while irredundancy needs windows of size 7. To check simultaneously these two properties, we will use windows of size 7. Such a window $(x_{i-3}, \dots, x_{i+3})$ will be used in the general case to check that x_i has a private neighbour and that every vertex of G in the interval $[x_{i-1}, x_i]$ is dominated. On the one hand the border window (x_1, \dots, x_6) will be used to prove that x_1, x_2 and x_3 have each a private neighbour and that all $v \prec x_3$ are dominated. On the other hand, the border window (x_{q-5}, \dots, x_q) will be used to prove that x_{q-2}, x_{q-1} and x_q have each a private neighbour and that all $v \succ x_{q-3}$ are dominated.

Therefore according to Lemma 35, Lemma 33 and Lemma 34, minimal dominating sets can be characterized through scanning windows of size 7 on ordered permutation graphs. More precisely, given a permutation graph $G = (V, E, \preceq)$ and $X = \{x_1 \preceq \dots \preceq x_q\} \subseteq V$, $X \in \text{MDS}(G)$ if and only if

- for every $(x_{i-3}, \dots, x_{i+3}) \in \mathbb{W}^7(X)$ we have
 - (degree) $\deg(\{x_{i-3}, \dots, x_{i+2}\}) \leq 2$
 - (irredundancy) $P_{sw_X^3(x_i)}(x_i) \neq \emptyset$
 - (domination) $\forall x_{i-1} \prec v \prec x_i, v \in N(sw_X^3(v))$
- for $(x_1, \dots, x_6) \in \mathbb{W}^6(X)$ we have
 - (irredundancy) $\bigwedge_{j \in \{1,2,3\}} P_{sw_X^3(x_j)}(x_j) \neq \emptyset$
 - (domination) $\forall v \prec x_3, v \in N(sw_X^3(v))$
- for $(x_{q-5}, \dots, x_q) \in \mathbb{W}^6(X)$ we have
 - (degree) $\deg(\{x_{q-5}, \dots, x_q\}) \leq 2$
 - (irredundancy) $\bigwedge_{j \in \{q-2, q-1, q\}} P_{sw_X^3(x_j)}(x_j) \neq \emptyset$

$$(\text{domination}) \forall v \succ x_{q-3}, v \in N(sw_X^3(v))$$

As in the case of interval graphs, the characteristic property that allows us to define locally basic properties, namely the bounded degree condition, could be omitted. Indeed one can prove that the local irredundancy condition implies that $\deg(X) \leq 2$.

We already know that MDS is a polynomial property, and with the characterization above we also know that it is locally definable on the class of permutation graphs. Notice that the formulas above are all decidable in polynomial time too, thus MDS is locally definable on the class of permutation graphs in polynomial time. Let us show that it is locally definable in time $O(n^8)$. We analyse the time to compute all ordered 6-tuples and 7-tuples of vertices satisfying the conditions of bounded degree, irredundancy and domination above. There are $O(n^7)$ such tuples. The condition on the bounded degree can be checked in constant time. Verifying the irredundancy of a vertex x_i consists in checking for each $v \in N[x_i]$ if v does not intersect any other element of the surrounding-window of x_i beside x_i itself. For each $v \in N[x_i]$ we just have to compare $t(v)$ and $b(v)$ to $t(x_j)$ and $b(x_j)$ for $j \in \{i-3, \dots, i+3\} \setminus i$, which takes a constant time. Thus in total we need a time in $O(n)$. We can proceed in the same way for the domination condition. For each v , $x_{i-1} \prec v \prec x_i$ we compare the top and bottom of v to the tops and bottoms of the elements of the surrounding-windows of v . This takes a time in $O(n)$ too. We conclude that ordered 6-tuples and 7-tuples satisfying the conditions of irredundancy and domination above can be all computed in time $O(n^8)$.

Hence, property MDS is locally definable in time $O(n^8)$ on the class of permutation graphs. Moreover, there are $O(n^6)$ sets X of vertices of size less than 7 that are candidates to be MDS. Verifying whether each of them is a minimal dominating sets can be done in linear time. So, dealing with minimal dominating sets of size less than 7 requires time $O(n^7)$. We conclude that `ENUM_MDS` is enumerable with linear delay after a pre-processing in $O(n^8)$. □

3.5.3 Permutation graphs and minimal connected dominating sets

Let us recall that MCDS states for the vertex set property that associates to each graph the collection of its maximal irredundant sets. We prove that maximal irredundant sets are efficiently enumerable for permutation graphs.

Proposition 37. *For the class of permutation graphs, `ENUM_MCDS` is enumerable with linear delay after a pre-processing in $O(n^7)$.*

Proof. We prove that MCDS is locally definable in polynomial time on permutation graphs; thus the above proposition follows from Theorem 22.

By the definition of minimal connected dominating sets in a permutation graph G , and since every such set has degree at most two (Lemma 35), $X \in \text{MCDS}(G)$ if and only if the following conditions hold:

- $\deg(X) \leq 2$
- X is connected
- every vertex $v \in V \setminus X$ is dominated by X
- for all $y \in X$, either $P_X(y) \neq \emptyset$ or $X \setminus \{y\}$ is disconnected.

Recall that the condition on the degree can be checked locally through a window of size 6 (Lemma 33). Under this degree assumption, Lemma 34 guarantees the local definability of connectivity, domination and private neighbourhood through windows of size at most 7.

Now notice the following: if deleting x_i from X breaks connectivity, then this is seeable on the windows of size 4 of $X \setminus x_i$, which are not windows of X . These windows are constituted only of elements of $sw_X^3(x_i)$. Hence, given $x_i \in X$ with $(x_{i-3}, \dots, x_{i+3}) \in \mathbb{W}^7(X)$, the set $X \setminus x_i$ is no longer connected if and only if

- $\deg(X) \leq 2$
- either $(\{x_{i-3}, x_{i-2}\} \cap N(x_{i-1}, x_{i+1}) = \emptyset)$
or $(\{x_{i-2}, x_{i-1}\} \cap N(x_{i+1}, x_{i+2}) = \emptyset)$
or $(\{x_{i-1}, x_{i+1}\} \cap N(x_{i+2}, x_{i+3}) = \emptyset)$

We conclude that we will need windows of size 7 in order to characterize minimal connected dominating sets. A general window $(x_{i-3}, \dots, x_{i+3})$ will be used to check the connectivity between x_{i-2} and x_{i-1} . This introduces a slight dissymmetry, thus the extremal window (x_1, \dots, x_6) will be used to check the connectivity between x_1 and x_2 and the extremal window (x_{q-5}, \dots, x_q) will be used to check the connectivity of $\{x_{q-4}, \dots, x_q\}$.

Observe that if X is connected, then all vertices $y \in V$ such that $x_1 \prec y \prec x_q$ are dominated. Indeed, if $y \in V$ is a vertex between x_i and x_{i+1} two successive elements of X , then according to Lemma 34 (Connectivity) and Lemma 32, y intersects x_{i-1}, x_i, x_{i+1} or x_{i+2} . Thus, if X is connected, in order to check if X is a dominating set, we need only to verify that each $y \prec x_1$ and each $y \succ x_q$ is dominated. This will be done through border windows (x_1, \dots, x_6) and (x_{q-5}, \dots, x_q) .

Now, it remains to deal with minimality. By the removal of one element either we loose the connectivity or we keep the connectivity and loose the domination. Like for interval graphs, keeping connectivity while loosing domination cannot happen during the removal of a vertex between x_1 and x_q . Hence, through a general window $(x_{i-3}, \dots, x_{i+3})$ we will ensure that $X \setminus x_i$ is no longer connected. Again there is a slight dissymmetry, extremal window (x_1, \dots, x_6) will be used to verify the minimality on $X \setminus x_1, X \setminus x_2, X \setminus x_3$ and extremal window (x_{q-5}, \dots, x_q) will be used to verify the minimality on $X \setminus x_{q-2}, X \setminus x_{q-1}, X \setminus x_q$. Verifying the minimality on $X \setminus x_2, X \setminus x_3$ and $X \setminus x_{q-2}, X \setminus x_{q-1}$ consists in making sure that each of these sets is disconnected. For $X \setminus x_1$ and $X \setminus x_q$ it is slightly different, not only domination but also connectivity can be lost by the removal of these extremal elements. Hence in addition, through border window (x_1, \dots, x_6) we need to check that $X \setminus x_1$ is either disconnected or that $P_X(x_1) \neq \emptyset$. Similarly through border window (x_{q-5}, \dots, x_q) we need to check that $X \setminus x_q$ is either disconnected or that $P_X(x_q) \neq \emptyset$.

Therefore according to these observations, to Lemma 35, Lemma 33 and Lemma 34, minimal connected dominating sets can be characterized through scanning windows of size 7 in ordered permutation graphs. More precisely, given a permutation graph $G = (V, E, \preceq)$ and $X = \{x_1, \dots, x_q\} \subseteq V$, $X \in \text{MCDS}(G)$ if and only if

- for every $(x_{i-3}, \dots, x_{i+3}) \in \mathbb{W}^7(X)$ we have
 - (degree) $\deg(\{x_{i-3}, \dots, x_{i+2}\}) \leq 2$
 - (connectivity) $\{x_{i-3}, x_{i-2}\} \cap N(x_{i-1}, x_i) \neq \emptyset$
 - (minimality)
 - either $(\{x_{i-3}, x_{i-2}\} \cap N(x_{i-1}, x_{i+1}) = \emptyset)$
 - or $(\{x_{i-2}, x_{i-1}\} \cap N(x_{i+1}, x_{i+2}) = \emptyset)$
 - or $(\{x_{i-1}, x_{i+1}\} \cap N(x_{i+2}, x_{i+3}) = \emptyset)$

- for $(x_1, \dots, x_6) \in \mathbb{W}^6(X)$ we have
 - (connectivity) $x_1 \in N(x_2, x_3)$
 - (domination) for all $a \prec x_1$, $a \in N(sw_X^3(a))$
 - (minimality)
 - $(\{x_2\} \cap N(x_3, x_4) = \emptyset \text{ or } P_{sw_X^3}(x_1) \neq \emptyset)$
 - and $(\{x_1, x_3\} \cap N(x_4, x_5) = \emptyset)$
 - and $(\{x_1, x_2\} \cap N(x_4, x_5) = \emptyset \text{ or } \{x_2, x_4\} \cap N(x_5, x_6) = \emptyset)$
- for $(x_{q-5}, \dots, x_q) \in \mathbb{W}^6(X)$ we have
 - (degree) $\deg(\{x_{q-5}, \dots, x_q\}) \leq 2$
 - (connectivity) $\bigwedge_{j \in \{q-5, q-4, q-3\}} \{x_j, x_{j+1}\} \cap N(x_{j+2}, x_{j+3}) \neq \emptyset$
and $x_q \in N(x_{q-1}, x_{q-2})$
 - (domination) for all $a \succ x_q$, $a \in N(sw_X^3(a))$
 - (minimality)
 - $(\{x_{q-1}\} \cap N(x_{q-2}, x_{q-3}) = \emptyset \text{ or } P_{sw_X^3}(x_q) \neq \emptyset)$
 - and $(\{x_{q-4}, x_{q-3}\} \cap N(x_{q-2}, x_q) = \emptyset)$
 - and $(\{x_{q-5}, x_{q-4}\} \cap N(x_{q-3}, x_{q-1}) = \emptyset \text{ or } \{x_{q-4}, x_{q-3}\} \cap N(x_{q-1}, x_q) = \emptyset)$

The complexity analysis of the pre-processing step for the enumeration of MCDS is analogous to the one made in the case of the enumeration of MDS. We can decide in polynomial time if a subset of vertices is a minimal connected dominating set or not, so MCDS is a polynomial property. The characterization above shows that MCDS is locally definable on the class of permutation graphs. There are $O(n^7)$ tuples of size 6 or 7 satisfying the conditions of minimal connected domination above. The conditions of bounded degree, connectivity and minimality that have to be verified on windows of size 7 can all be checked in constant time. For extremal windows of size 6, they take a linear time. Hence 6-tuples and 7-tuples satisfying the conditions of minimal connected domination above can be all computed in time $O(n^7)$.

Property MCDS is locally definable in time $O(n^7)$ on the class of permutation graphs. Moreover, there are $O(n^6)$ sets X vertices of size less than 7 that are candidates to be MCDS. Verifying whether each of them is a minimal connected dominating set can be done in linear time. So, dealing with minimal dominating sets of size less than 7 requires time $O(n^7)$. We conclude that ENUM_MCDS is enumerable with linear delay after a pre-processing in $O(n^7)$.

□

3.5.4 Permutation graphs and maximal irredundant sets

We recall that MIR stands for the vertex set property that associates to each graph the collection of its maximal irredundant sets. We prove that maximal irredundant sets are efficiently enumerable on permutation graphs.

Proposition 38. *For the class of permutation graphs, ENUM_MIR is enumerable with linear delay after a pre-processing in $O(n^{13})$.*

Proof. We prove that MIR is polynomial time locally definable on the class of permutation graphs. Thus the above proposition follows from Theorem 22.

Let $G = (V, E)$ be such an ordered permutation graph and $X \subseteq V$. Recall that set X is a maximal irredundant set if:

- (irredundancy) for all $y \in X$, $P_X(y) \neq \emptyset$,
- (maximality) for all $a \in V \setminus X$ either $P_{X \cup a}(a) = \emptyset$ or $P_{X \cup a}(s) = \emptyset$ for some $s \in X$.

By Lemma 35 a maximal irredundant set X does not contain any vertex v such that $\deg_X(v) > 2$, and this can be verified locally by Lemma 33. As a consequence, by Lemma 34, one benefits from the local characterization of the private neighbourhood and thus definition above is equivalent to:

The set X is a maximal irredundant set if and only if:

- $\deg(X) \leq 2$
- (irredundancy) for all $y \in X$, $P_X(y) \neq \emptyset$,
- (maximality) for all $a \in V \setminus X$,

either $\deg(X \cup \{a\}) > 2$,

or $\deg(X \cup \{a\}) \leq 2$ and

either $P_{X \cup a}(a) = \emptyset$

or $P_{X \cup a}(s) = \emptyset$ for some $s \in X$.

According to Lemma 33 one can verify locally that $\deg(X) \leq 2$. Under this condition one has a local characterization of private neighbourhood, see Lemma 34. Moreover, when $\deg(X) \leq 2$, if $X \cup a$ contains a vertex v with $\deg_X(v) > 2$, then this is detectable through the windows of size 6 of $X \cup a$ containing a that are not windows of X . These windows are covered by $sw_{X \cup a}^5(a)$.

It remains to check that an s in X such that $P_X(s) \neq \emptyset$ and $P_{X \cup a}(s) = \emptyset$ can be found locally. Actually the only vertices whose private neighbourhood is affected by the addition of a to X are a itself and the vertices at X -distance at most 3 to a . Therefore such s belongs to $sw_X^3(a)$. For each such candidate we need to check (locally) its private neighbourhood. Therefore a window of size 12 is needed to check maximality, see Figure 3.20 below.

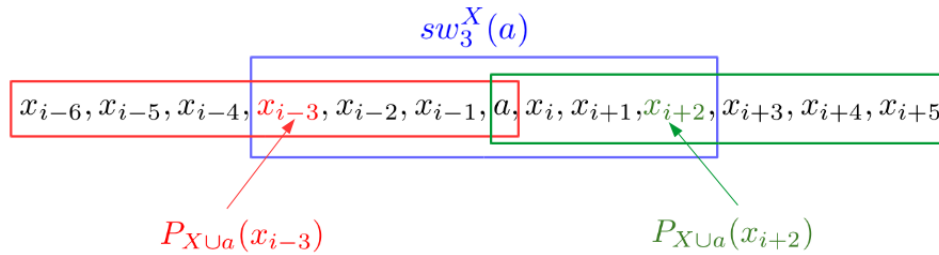


Figure 3.20: Expressing maximality through 12-windows

As a consequence, a maximal irredundant set can be entirely defined locally on permutation graphs. Irredundancy is locally checked through windows of size 7 and maximality through windows of size 12. The property of being maximal irredundant will thus be checked through windows of size 12. On window $(x_{i-6}, \dots, x_{i+5})$ the irredundancy will be checked on x_i and the maximality will be checked between x_{i-1} and x_i . On border window (x_1, \dots, x_{11}) we will ensure that $\{x_1, \dots, x_6\}$ is irredundant and that the maximality is respected on the left of x_6 . Finally on border window (x_{q-10}, \dots, x_q) we will ensure that $\{x_{q-4}, \dots, x_q\}$ is irredundant and that the maximality is respected on the right of x_{q-5} .

From these observations, from Lemma 35, Lemma 33 and Lemma 34, we get that $X \in \text{MIR}(G)$ if and only if:

- for every $(x_{i-6}, \dots, x_{i+5}) \in \mathbb{W}^{12}(X)$ we have
 - (degree) $\deg(\{x_{i-6}, \dots, x_{i-1}\}) \leq 2$
 - (irredundancy) $P_{sw_X^3(x_i)}(x_i) \neq \emptyset$
 - (maximality) for all $a \in V \setminus X$ such that $x_{i-1} \prec a \prec x_i$,
 - either: $\deg(sw_{X \cup a}^5(a)) > 2$
 - or: $P_{sw_{X \cup a}^3(a)}(a) = \emptyset$ or there exists $x_j \in sw_{X \cup a}^3(a)$ such that $P_{sw_{X \cup a}^3(x_j)}(x_j) = \emptyset$
- for $(x_1, \dots, x_{11}) \in \mathbb{W}^{11}(X)$ we have
 - (irredundancy) $\bigwedge_{j \in \{1, \dots, 6\}} P_{sw_X^3(x_j)}(x_j) \neq \emptyset$
 - (maximality) for all $a \in V \setminus X$ such that $a \prec x_6$, same condition as for the general window.
- for $(x_{q-10}, \dots, x_q) \in \mathbb{W}^{11}(X)$ we have
 - (degree) $\bigwedge_{j \in \{q-10, \dots, q-5\}} \deg(\{x_j, \dots, x_{j+5}\})$
 - (irredundancy) $\bigwedge_{j \in \{q-4, \dots, q\}} P_{sw_X^3(x_j)}(x_j) \neq \emptyset$
 - (maximality) for all $a \in V \setminus X$ such that $a \succ x_{q-5}$, same condition as for the general window.

We already know that MIR is a polynomial property, and with the characterization above we know that it is also locally definable on the class of permutation graphs in polynomial time. Let us show that it is locally definable in time $O(n^{13})$.

We analyse the time to compute all 11-tuples and 12-tuples satisfying the conditions of maximal irredundancy above. There are $O(n^{12})$ such tuples. Observe once more that an analysis through the tops and bottoms enables us to check the degree of a window in constant time. We have seen in the complexity analysis of MDS that irredundancy can be checked in $O(n)$. Hence, verifying the maximality takes a time in $O(n^2)$ since the private neighbourhoods have to be computed for all a . This would give us a total time in $O(n^{14})$ but we can do better by adding a pre-processing step.

If we pre-calculate in a table \mathbb{T} the set $P_Z(z)$ for all ordered set Z of size at most 7 and all vertex z in Z , verifying maximality would reduce to $O(n)$. The time needed to compute this table is in $O(n^8)$ and hence is negligible in comparison to the number of tuples to consider. We conclude that tuples of size 11 and 12 satisfying the conditions of maximal irredundancy above can be all computed in time $O(n^{13})$.

Property MIR is locally definable in time $O(n^{13})$ on the class of permutation graphs. Moreover, there are n^{11} sets X of vertices such that $|X| < 12$ that are candidates to be MIR. Verifying whether each of them is a maximal irredundant sets can be done in linear time. So, dealing with maximal irredundant sets of size less than 12 requires time $O(n^{12})$. We conclude that ENUM_MIR is enumerable with linear delay after a pre-processing in $O(n^{13})$. □

As in the case of interval graphs, let us conclude this section by the following interesting corollary.

Corollary 39. *On permutation graphs the problem of finding a minimum connected dominating set, as well as the problem of finding a maximum irredundant set can be solved in polynomial time. Moreover, it is possible to count the minimal connected dominating sets and the maximal irredundant sets in polynomial time on such graphs.*

3.6 Local properties and enumeration for cyclically ordered graphs

A *cyclic order* is a way to arrange clockwise a set of objects in a circle. Formally, it is a ternary relation $[xyz]$, which means that after x one reaches y before z , and which fulfills the following axioms.

$$\begin{array}{ll} (\text{co}_1) [xyz] \Rightarrow [yzx]; & (\text{co}_3) ([xyz] \text{ and } [yzt]) \Rightarrow [xzt];^2 \\ (\text{co}_2) [xyz] \Rightarrow \neg[zyx]; & (\text{co}_4) [xyz] \text{ or } [zyx] \text{ for any p.w.d. } x, y, z. \end{array}$$

Given $n > 3$ points in a cyclically ordered set X , we write $[x_1x_2 \dots x_q]$ when $[x_ix_jx_k]$ holds for any $1 \leq i < j < k \leq q$. For any $x \neq y$ in X , we say that y is the *successor* of x (or that x is the *predecessor* of y) if there is no $a \in X$ satisfying $[xay]$. Clearly, each element of X has one predecessor and one successor.

As a consequence Definition 19 still has a precise meaning in the present framework. A k -window of X in a cyclically ordered graph G is a tuple (x_1, \dots, x_k) such that x_{i+1} is the successor of x_i for each $i < k$. For example, if $[x_1x_2 \dots x_q]$ holds, the 3-windows of the cyclically ordered set $\{x_1x_2 \dots x_q\}$ are the tuples

$$(x_1, x_2, x_3), (x_2, x_3, x_4), \dots, (x_{q-2}, x_{q-1}, x_q), (x_{q-1}, x_q, x_1), (x_q, x_1, x_2).$$

The set of k -windows in a cyclically ordered set is still denoted by $\mathbb{W}^k(X)$. The underlying cyclic order will always be clear from the context.

As in the previous sections, we are interested in locally definable vertex set properties. In the case of cyclic orders, this notion is more uniform than in the linear order case since there are no extremal cases to consider.

Definition 40. *Let \mathcal{G} be a class of graphs equipped with a uniform cyclic order. A vertex set property \mathcal{P} is locally definable on \mathcal{G} if there exist an integer $k > 1$ and a uniform vertex constraint ϕ of arity k , such that for any $G \in \mathcal{G}$ and any $X \subseteq V(G)$ of cardinality $> k$:*

$$X \in \mathcal{P}_G \text{ iff } \forall (a_1, \dots, a_k) \in \mathbb{W}^k(X), \phi_G(a_1, \dots, a_k).$$

If furthermore ϕ is decidable in polynomial time, then \mathcal{P} is said to be locally definable in polynomial time.

This definition is illustrated in the case where $k = 3$ in Figure 3.21.

²This axiom is often written under the equivalent form: $([xyz] \text{ and } [xzt]) \Rightarrow [xyt]$.

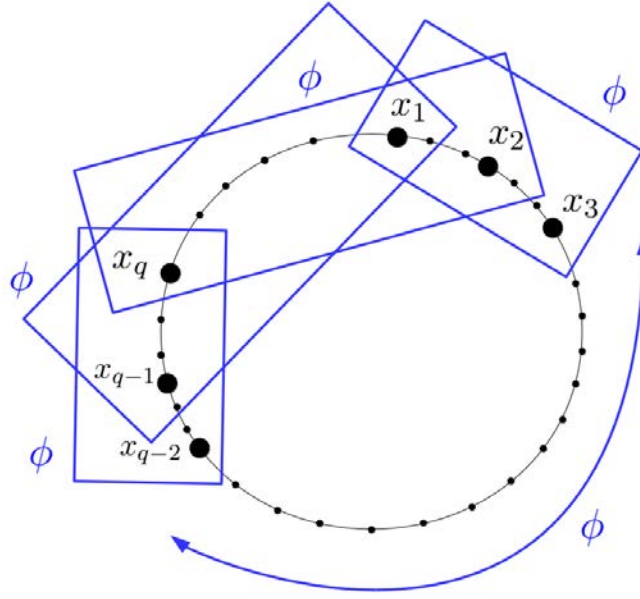


Figure 3.21: Locally definable properties through 3-windows in cyclically ordered graphs

The rest of this section is devoted to proving an analogue of Theorem 22. We establish that vertex set properties locally definable on a class of cyclically ordered graphs are enumerable with linear delay on this class. As previously this is done by reduction to the enumeration of paths in a DAG, but the reduction is more involved. Indeed, we will use a Turing reduction, which means that we will not make only one call to the enumeration procedure of paths in a DAG but a polynomial number of such calls.

Theorem 41. *Let \mathcal{G} be a class of cyclically ordered graphs and \mathcal{P} be a uniform vertex set property. If \mathcal{P} is locally definable in polynomial time on \mathcal{G} , then \mathcal{P} is enumerable with linear delay.*

Proof. We reduce the problem to that of enumeration of paths in a DAG. With each cyclically ordered graph $G = (V, E)$ in \mathcal{G} , we associate a new directed graph G_Δ^α in choosing a new point α on the circle and in setting $G_\Delta^\alpha = (V_\Delta^\alpha, E_\Delta^\alpha)$, where:

$$V_\Delta^\alpha = \{(x_1, \dots, x_{k-1}) \in V^{k-1} \text{ s.t. } [x_1 \dots x_{k-1} \alpha]\}$$

and a pair of vertices $(x_1, \dots, x_{k-1}), (y_1, \dots, y_{k-1}) \in V_\Delta^\alpha$ is an arc of E_Δ^α if

$$(x_2, \dots, x_{k-1}) = (y_1, \dots, y_{k-2}) \text{ and } \phi(x_1, \dots, x_{k-1}, y_{k-1}).$$

Notice that G_Δ^α is a DAG since for any path in G_Δ^α , say:

$$(x_1, x_2, \dots, x_{k-1}) \rightarrow (x_2, x_3, \dots, x_k) \rightarrow \dots \rightarrow (x_p, x_{p+1}, \dots, x_{p+k-2}),$$

the definition of V_Δ^α implies that $[x_i \dots x_{i+k-2} \alpha]$ holds for each $i \leq p$. In particular, we have $[x_1 x_2 \alpha]$, $[x_2 x_3 \alpha]$, \dots , $[x_{p-1} x_p \alpha]$ and hence $[x_1 x_{p-1} \alpha]$, by (co₃). Therefore, the equality $(x_p, \dots, x_{p+k-2}) = (x_1, \dots, x_{k-1})$ cannot hold, since it would entail $x_p = x_1$ and hence the assertions $[x_{p-1} x_p \alpha]$ and $[x_1 x_{p-1} \alpha]$ stated above could be rephrased into $[x_{p-1} x_1 \alpha]$ and $[x_1 x_{p-1} \alpha]$, in contradiction with (co₂). Thus, there is no directed cycle in G_Δ^α .

As a consequence of Definition 40 and of the definition of G_Δ^α , each set $\{x_1, \dots, x_q\} \in \mathcal{P}_G$ (enumerated in such a way that $[x_1 \dots x_q \alpha]$ holds) defines a path $(x_1, \dots, x_{k-1}) \rightarrow (x_2, \dots, x_k) \rightarrow \dots \rightarrow (x_{q-k+2}, \dots, x_q)$

in G_Δ^α . Observe that the converse is false. Indeed, the existence of a path $(x_1, \dots, x_{k-1}) \rightarrow \dots \rightarrow (x_{q-k+2}, \dots, x_q)$ in G_Δ^α guarantees that the requirements $\phi(x_1, \dots, x_k), \dots, \phi(x_{q-k+1}, \dots, x_q)$ are fulfilled, but it does not certify the satisfaction of the $k - 1$ additional constraints yet mandatory to ensure that $\{x_1, \dots, x_q\} \in \mathcal{P}_G$, namely:

$$\phi(x_{q-k+2}, \dots, x_q, x_1), \phi(x_{q-k+3}, \dots, x_q, x_1, x_2), \dots, \phi(x_q, x_1, x_2, \dots, x_{k-1}) \quad (3.2)$$

These $k - 1$ additional constraints are highlighted in Figure 3.22.

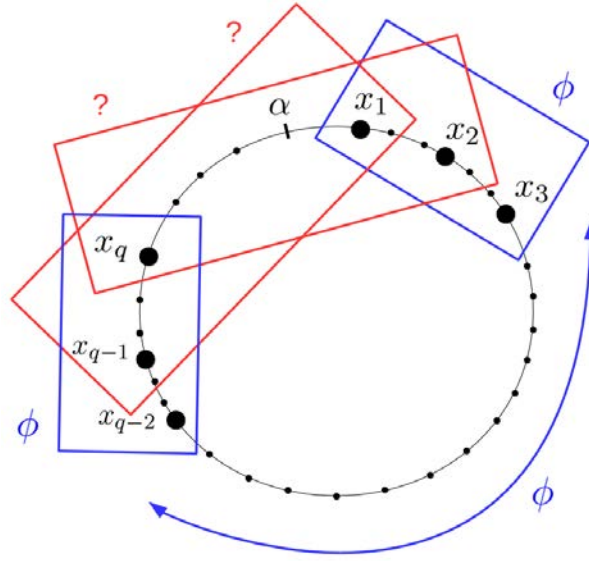


Figure 3.22: Additional constraints to ensure for $k = 3$

In other words, the sets of vertices supporting a path in G_Δ^α include all sets of \mathcal{P}_G plus some sets out of \mathcal{P}_G . So we must select among the paths in G_Δ^α those that do correspond to some set of \mathcal{P}_G .

In order to do so we identify all pairs of vertices $(s, t) \in V_\Delta^\alpha \times V_\Delta^\alpha$ that are endpoints of a “good” path in G_Δ^α , that is, a path whose vertex support does belong to \mathcal{P}_G . Such pairs (s, t) in $V_\Delta^\alpha \times V_\Delta^\alpha$ fulfill the following conditions (see Figure 3.23 for an illustration):

- (i) there is at least one s, t -path in G_Δ^α ;
- (ii) (s, t) satisfies (3.2), that is with $s = (x_1, \dots, x_{k-1})$ and $t = (y_1, \dots, y_{k-1})$:

$$\phi(y_1, \dots, y_{k-1}, x_1), \phi(y_2, \dots, y_{k-1}, x_1, x_2), \dots, \phi(y_{k-1}, x_1, x_2, \dots, x_{k-1}).$$

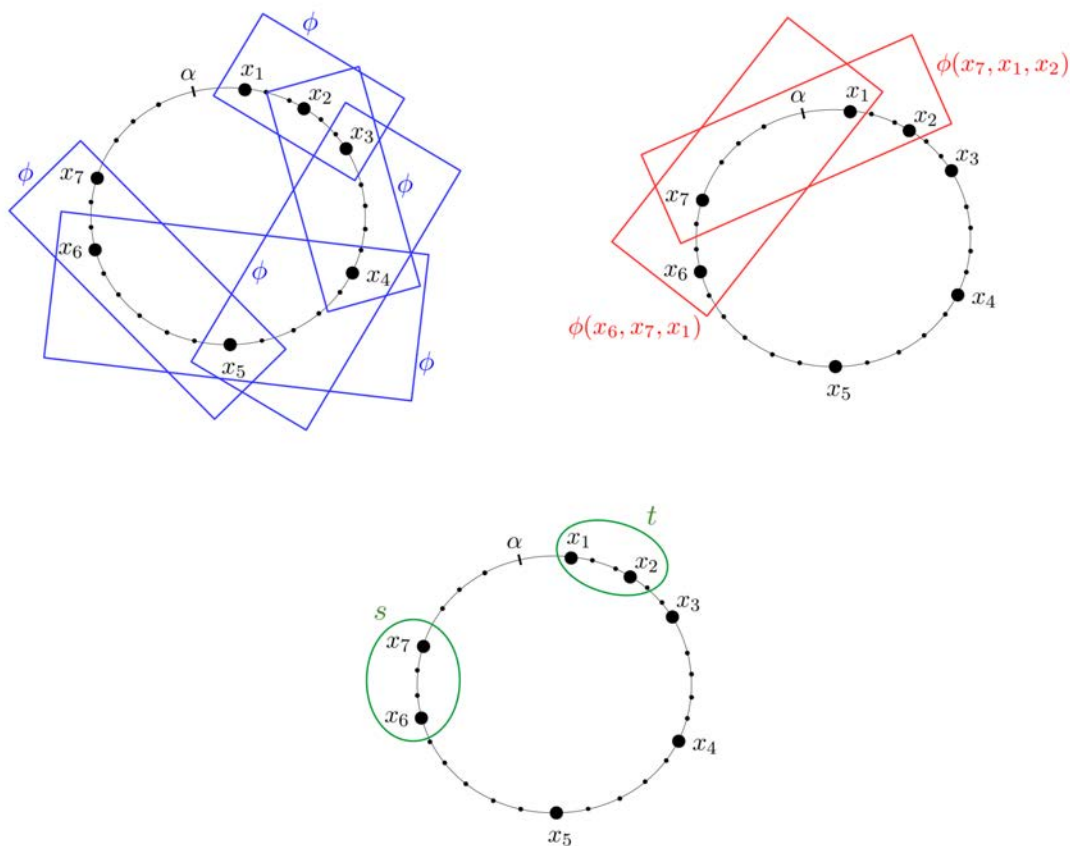


Figure 3.23: Endpoints of path (x_1, \dots, x_7) in G_{Δ}^{α} for $k = 3$

By construction, there is a one-to-one correspondence between the sets of \mathcal{P}_G and the s, t -paths in G_{Δ}^{α} whose pair of endpoints (s, t) belongs to the following set $EP^{\alpha}(G)$:

$$EP^{\alpha}(G) = \{(s, t) \in V_{\Delta}^{\alpha} \times V_{\Delta}^{\alpha} \text{ s.t. } s \text{ and } t \text{ fulfill conditions (i) and (ii) above}\}.$$

All in all, we justified the following enumeration algorithm of \mathcal{P}_G :

<p>Algorithm 13: An algorithm for $ENUM_{\mathcal{P}}$ for cyclically ordered graphs</p> <p>Input: a cyclic ordered graph G Output: an enumeration of \mathcal{P}_G</p> <ol style="list-style-type: none"> 1 Choose a new point α on the circle 2 Construct G_{Δ}^{α} 3 Construct $EP^{\alpha}(G)$ 4 $L \leftarrow \emptyset$ 5 for each $(s, t) \in EP^{\alpha}(G)$ do 6 $H \leftarrow CLEAR_{DAG}(G_{\Delta}^{\alpha}, s, t)$ 7 $L \leftarrow L + (H, s, t)$ 8 for each $(H, s, t) \in L$ do 9 $ENUM_PATHS_{DAG}(H, s, t)$

Let us examine its complexity. Let n denote the number of vertices of G . Assume ϕ can be decided in time $O(n^c)$. Constructing $G_\Delta^\alpha = (V_\Delta^\alpha, E_\Delta^\alpha)$ in Line 2 takes time $O(n^{k+c})$. When G_Δ^α is computed, EP^α is built by iterative inclusion of all couples $(s, t) \in V_\Delta^\alpha \times V_\Delta^\alpha$ that fulfill conditions (i) and (ii) above. Checking (i) means testing reachability between s and t in G_Δ^α . It takes time $O(|V_\Delta^\alpha| + |E_\Delta^\alpha|) = O(n^k)$. For (ii), one must verify whether ϕ holds for $k-1$ tuples: it takes time $O(n^c)$. Hence, Line 3 runs in time $O(n^{2k-2})(O(n^k) + O(n^c)) = O(n^{3k-2} + n^{2k-2+c})$.

Once EP^α are available, we know all the couples (s, t) that are the endpoints of paths in G_Δ^α that really correspond to some set $X \in \mathcal{P}_G$. In order to enumerate these paths without wasting time in vain explorations, we must nevertheless “clean” G_Δ^α with respect to these couples of endpoints. Lines 5 to 7 are devoted to this task. The list L collects all triples (H, s, t) , where s, t are the endpoints of an “authorized” path in G_Δ^α , and H is the graph G_Δ^α cleaned from all vertices that do not lead to t . The time complexity of this whole step is bounded by $|\text{EP}^\alpha(G)|O(|V_\Delta^\alpha| + |E_\Delta^\alpha|) = O(n^{2k-2})O(n^k) = O(n^{3k-2})$.

As a consequence, Lines 1 to 7 of the algorithm, which correspond to the pre-processing step of the enumeration, consumes a time $O(n^{3k-2} + n^{2k-2+c})$.

After this pre-processing step, the enumeration in the strict sense is performed in Lines 8 and 9. On the one hand, observe that every call to the procedure `ENUM_PATHSDAG` leads to at least one solution by the choice of $\text{EP}^\alpha(G)$. On the other hand note that the size of the paths outputted is linear in the size of the vertex sets they represent. Therefore, thanks to Theorem 20, the enumeration runs with linear delay. \square

Remark 2. *The above algorithm enumerates sets of size $\geq k$ only. Nevertheless, it can be easily extended so that it enumerates sets of all cardinalities. Indeed, there are at most $O(n^{k-1})$ sets of size $< k$, and for such a set X one can decide in polynomial time whether X belongs to \mathcal{P}_G . Therefore a list of the sets of \mathcal{P}_G of size less than k can be built within the pre-processing step, and they can be later enumerated from this list at the beginning of the enumeration phase. For all the vertex set properties \mathcal{P} considered in this chapter, one can decide whether a fixed size set X belongs to \mathcal{P}_G in time $O(n)$. Therefore, the consideration of all these sets requires a total time $O(n^k)$ and does not increase the complexity of the pre-processing step. From now on, we will not mention the specific treatment of these “small” vertex sets, since it does not affect our results.*

3.7 MDS and MIR in circular-arc and circular-permutation graphs

We present the class of circular-arc graphs and give a characterization of minimal dominating sets and maximal irredundant sets in this class of graphs. The same work is then done for the class of circular-permutation graphs.

3.7.1 Enumeration in circular-arc graphs

As a first example of cyclically ordered graph class, we consider the class of circular-arc graphs, which is a subclass of intersection graphs. The intersection model of a circular-arc graph is composed of arcs on a circle. If G is such a graph, we denote by $[s(x), e(x)]$ the arc associated with a vertex x and we define such an arc as the set of all points u on the circle such that $[s(x), u, e(x)]$, with $s(x)$ and $e(x)$ respectively the beginning and the end of x following clockwise the circle. We assume without loss of generality that all endpoints are pairwise distinct. The natural cyclic order associated with a circular-arc graph is the following: $[x, y, z]$ if and only if $[s(x), s(y), s(z)]$, which means that after $s(x)$ one reaches $s(y)$ before $s(z)$ when following clockwise the circle.

An example of circular-arc graph and its intersection model is given in Figures 3.24 and 3.25.

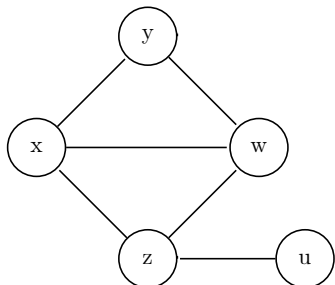


Figure 3.24: Circular-arc graph G

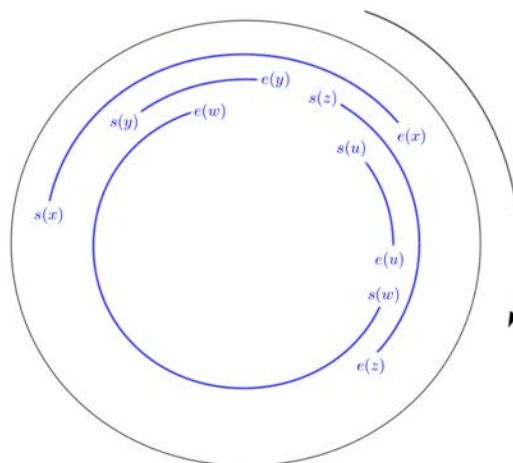


Figure 3.25: Intersection model of G

Some properties locally definable on interval graphs can be defined very similarly on circular-arc graphs. Given a set of vertices X of such a graph, we say that X has no nested circular-arcs if for all distinct vertices x, y in X , $[s(y), e(x), e(y)]$ whenever $[s(x), s(y), e(x)]$.

Under this condition it is easy to check that the properties of domination and private neighbourhood can be locally defined. The proofs of the two lemmas below are very similar to the proofs of Lemma 24 and Lemma 25, the only difference is that there are no extremal windows to consider.

Lemma 42. *Let $G = (V, E)$ be a circular-arc graph, and $X \subseteq V$. The set X contains no nested circular-arcs if and only if for all $(x, y) \in \mathbb{W}^2(X)$, $[s(y), e(x), e(y)]$ whenever $[s(x), s(y), e(x)]$.*

Lemma 43. *Let $G = (V, E)$ be a circular-arc graph, and $X \subseteq V$ with no nested circular-arcs. Then:*

(Domination) *For every $(x, y) \in \mathbb{W}^2(X)$ and every $v \notin X$ such that $[x, v, y]$:*

$$v \in N(X) \text{ iff } v \in N(x, y).$$

(Private neighbourhood) *For every $(x, y, z) \in \mathbb{W}^3(X)$: $P_X(y) = P_{\{x, y, z\}}(y)$.*

Interestingly, connectivity cannot be defined locally on the class of circular-arc graphs, contrary to interval graphs. Indeed consider for instance the set $X = \{x_1, \dots, x_{2n}\}$ such that $[s(x_{2n}), s(x_1), e(x_{2n})]$, and for all i , $[s(x_i), s(x_{i+1}), e(x_i)]$ except for $i = n$ where $[s(x_n), e(x_n), s(x_{n+1})]$ holds. The set X is connected, see Figure 3.26. Therefore, the fact $[s(x), s(y), e(x)]$ holds for all $(x, y) \in \mathbb{W}^2(X)$ is not necessary to have connectivity. Suppose now that $[s(x_{2n}), s(x_1), e(x_{2n})]$ does not hold, either, then X is not connected anymore. Intuitively this means that X has two diametrically opposite holes, see Figure 3.27. This cannot be detected through a fixed size window.

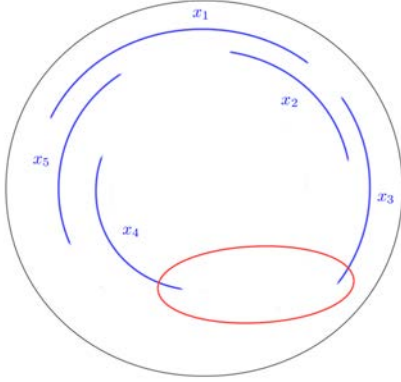


Figure 3.26: Connected circular-arc graph

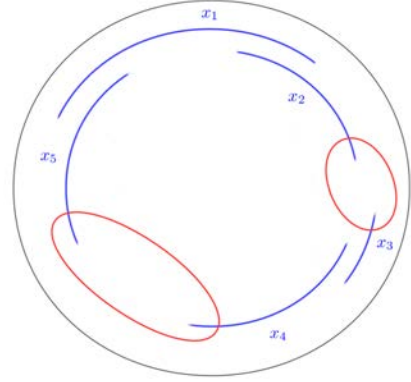


Figure 3.27: Disconnected circular-arc graph

Lemma 44. *An irredundant set of vertices in a circular-arc graph has no nested arcs.*

As in Section 3.4 the three lemmas above together with Theorem 41 enable us to efficiently enumerate minimal dominating sets and maximal irredundant sets on circular-arc graphs.

Proposition 45. *On circular-arc graphs, the minimal dominating sets are enumerable with linear delay after a polynomial pre-processing.*

Proof. According to Lemma 43 and Lemma 44, minimal dominating sets can be characterized through scanning windows of size 3 on ordered circular-arc graphs. More precisely, given such a graph G and $X \subseteq V$, $X \in \text{MDS}(G)$ if and only if for every $(x, y, z) \in \mathbb{W}^3(X)$:

(No nested arcs) if $[s(x), s(y), e(x)]$ then $[s(y), e(x), e(y)]$

(Irredundancy) $P_{\{x,y,z\}}(y) \neq \emptyset$;

(Domination) for any $a \in V \setminus X$ such that $[x, a, y]$, $a \in N(x, y)$.

As for interval graphs, the condition "no nested arcs" can be omitted since it follows from the fact that for every $(x, y, z) \in \mathbb{W}^3(X)$, $P_{\{x,y,z\}}(y) \neq \emptyset$ (irredundancy condition), therefore we can omit it.

This characterization shows that MDS is locally definable on the class of circular-arc graphs (see Definition 40). Therefore, according to Theorem 41 there is a linear delay enumeration algorithm for MDS on circular-arc graphs. \square

Proposition 46. *On circular-arc graphs, the maximal irredundant sets are enumerable with linear delay after a polynomial pre-processing.*

Proof. According to Lemma 43 and Lemma 44, maximal irredundant sets can be characterized through scanning windows of size 4 in ordered circular-arc graphs. More precisely, given such a graph G and $X \subseteq V$, $X \in \text{MIR}(G)$ if and only if for every $(w, x, y, z) \in \mathbb{W}^4(X)$:

(No nested arcs) if $[s(w), s(x), e(w)]$ then $[s(x), e(w), e(x)]$

(Irredundancy) $P_{\{w,x,y\}}(x) \neq \emptyset$;

(Maximality) for any $a \in V \setminus X$,

$$[x, a, y] \Rightarrow \begin{cases} ([s(x), s(a), e(x)] \text{ and } [s(a), e(a), e(x)]) \text{ or} \\ ([s(a), s(y), e(a)] \text{ and } [s(y), e(y), e(a)]) \text{ or} \\ P_{\{w,x,a\}}(x) = \emptyset \text{ or } P_{\{x,a,y\}}(a) = \emptyset \text{ or } P_{\{a,y,z\}}(y) = \emptyset \end{cases}$$

As for interval graphs, the condition "no nested arcs" can be omitted.

This characterization shows that MIR is locally definable on the class of circular-arc graphs (see Definition 40). Therefore, according to Theorem 41 there is a linear delay enumeration algorithm for MIR on circular-arc graphs. \square

3.7.2 Enumeration in circular-permutation graphs

A second example of cyclically ordered graph class is the class of circular-permutation graphs, which is also a subclass of intersection graphs. The intersection model of a circular-permutation graph is composed of straight lines between two concentric circles. We keep the same notations as in our section on permutation graphs and we order the vertices by their bottom line endpoints.

An example of circular-permutation graph and its intersection model is given in Figures 3.28 and 3.29.

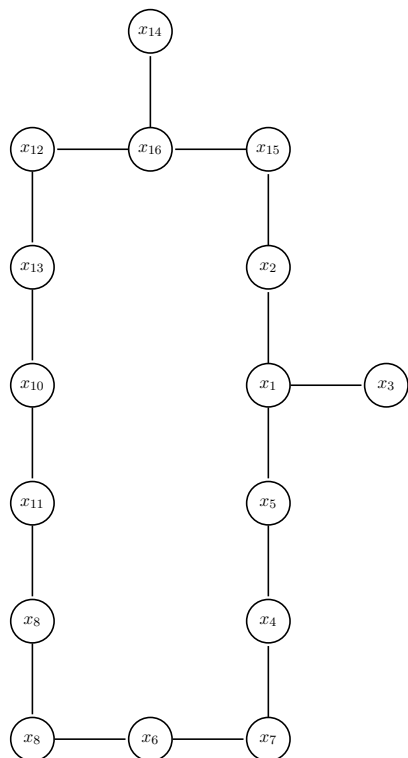


Figure 3.28: Circular-permutation graph G

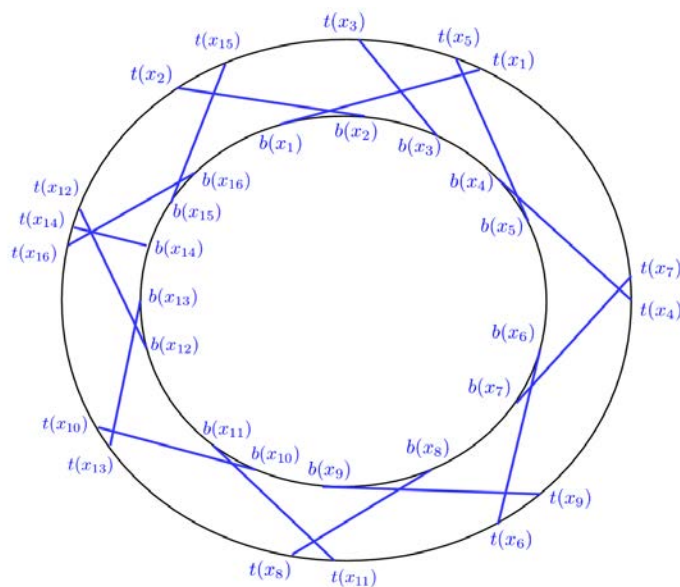


Figure 3.29: Intersection model of G

Some properties locally definable on the linearly ordered permutation graphs can be expressed on circular-permutation graphs very similarly. In particular Lemma 33, the second and third item of Lemma 34 and Lemma 35 hold as well for circular-permutation graphs. Therefore, according to Theorem 41 we have the following result.

Proposition 47. *On circular-permutation graphs, the minimal dominating sets and the maximal irredundant sets are enumerable with linear delay after a polynomial pre-processing.*

Observe that, connectivity cannot be defined locally on circular-permutation graphs for the same reason as on circular-arc graphs.

The proofs of Proposition 46 and Proposition 47 give the interesting following corollary, which answers to - as far as we know- open questions.

Corollary 48. *On circular-arc graphs, as well as on circular-permutation graphs, the problem of finding a maximum irredundant set can be solved in polynomial time. Moreover, it is possible to count the maximal irredundant sets in polynomial time on such graphs.*

Proof. Let us give the proof in the case of circular-arc graphs (the proof for circular-permutation graphs is similar). The property of maximal irredundant set is locally definable on circular-arc graphs through windows of size 7. We first list maximal irredundant sets of size less than 7. Second, we use the enumeration algorithm implicitly used in the proof of Proposition 46 and described in the proof of Theorem 41. In this algorithm we replace all calls to $\text{ENUM_PATHSDAG}(H, s, t)$ by calls to a polynomial-time procedure computing a longest path in H from s to t , and we add the irredundant sets corresponding to the paths so obtained to the above list. A maximum irredundant set belongs to this polynomial-size list of irredundant sets.

The proof for the counting problem is very similar. This time, in Algorithm 13, we replace all calls to $\text{ENUM_PATHSDAG}(H, s, t)$ by calls to the counting Algorithm 12. Finally, we make the sum of the result of each of those calls plus the number of maximal irredundant sets of size less than 7 listed before. \square

3.8 Conclusion

A first interesting result of this chapter comes from Theorem 15: we proposed a characterization of the graphs for which for each induced subgraph the sets of minimal dominating sets and maximal irredundant sets correspond. Thus, in the frame of the class of MDS-MIR perfect graphs, if we get an algorithm to solve the problem of enumerating all minimal dominating sets, we solve at the same time the problem of enumerating all maximal irredundant sets. It is for example possible to enumerate all minimal dominating set in total time $O(1.5705^n)$ in the class of cographs [23], or in total time $O(1.4423^n)$ in the class of split graphs [24]. On the class of split graphs, there also exists a polynomial delay algorithm for enumerating minimal dominating sets [64]. Cographs and split graphs being MDS-MIR perfect graphs, those enumeration results gives us also results on the generation of maximal irredundant sets.

A second interesting result is Theorem 22: we proposed a general method to find linear delay algorithms for the enumeration of vertex set properties on linearly ordered graphs. We proved that if the property is locally definable, then the corresponding enumeration problem reduces to the enumeration of paths in directed acyclic graphs. We illustrated this method on the class of interval graphs and on the class of permutation graphs. On such graphs we showed that the minimal connected dominating sets and the maximal irredundant sets can be enumerated with linear delay. In addition, as a corollary we obtained a polynomial time algorithm for finding a minimum connected dominating set and a maximum irredundant set, as well as a polynomial time algorithm for counting minimal connected dominating sets and maximal irredundant sets. It is worth noticing that our method, which highlights the local character of the studied properties, gives combinatorial proofs that are simpler than previously existing ones for minimal dominating sets in interval and permutation graphs [66].

In a second step we extended our results to cyclically ordered graphs. We obtained a similar general method for enumerating vertex set properties on cyclically ordered graphs with linear delay. We applied it to the enumeration of maximal irredundant sets in the class of circular-arc graphs and in the class of circular-permutation graphs. As a corollary we provided a polynomial time algorithm for finding a

maximum irredundant set and a polynomial time algorithm for counting maximal irredundant sets in such graphs.

A natural issue is to search for other classes of graphs and other enumeration problems to apply our method. A promising line of research is to proceed with the class of graphs of bounded linear induced matching width (see [52]) and to define in this context an appropriate variant of the local definability of vertex set properties.

Efficient enumeration of k -arc-connected orientations in graphs

Contents

4.1	Introduction	63
4.2	Preliminaries	67
4.2.1	Notations and basics of orientations	68
4.2.2	Frank's theorem on k -arc-connected orientations	69
4.3	Enumeration of k-connected orientations via submodular flows	74
4.3.1	Partial orientation extension with submodular flow feasibility	75
4.3.2	Partial orientation extension using minimum-cost orientation	77
4.4	Orientations with prescribed outdegree sequence	78
4.4.1	Flashlight backtrack search via integer points of polytopes	78
4.4.2	Flashlight backtrack search via cycle flips	88
4.5	k-connected outdegree sequences via path flips	90
4.5.1	Reverse search	93
4.5.2	Flashlight backtrack search via path flips	96
4.6	Simple and modular enumeration of k-connected orientations	99
4.7	Conclusion	102

We address here the second issue of the thesis. This work is independent of the previous one on vertex set properties done in the last chapter, it deals with the enumeration of k -arc-connected orientations in graphs.

4.1 Introduction

Given an undirected (not necessarily simple) graph G , we consider the problem of enumerating its orientations with given properties, i.e., outputting each orientation exactly once. The set of all orientations of $G = (V, E)$ can be identified with the set of vectors $\{0, 1\}^m$ and exploring the latter can be done using a gray code, thus by [30] enumerating all possible orientations of an undirected graph can be done with constant delay. It gets more interesting when enumerating *acyclic* orientations, i.e. those that have no directed cycles, e.g., Figure 4.1.

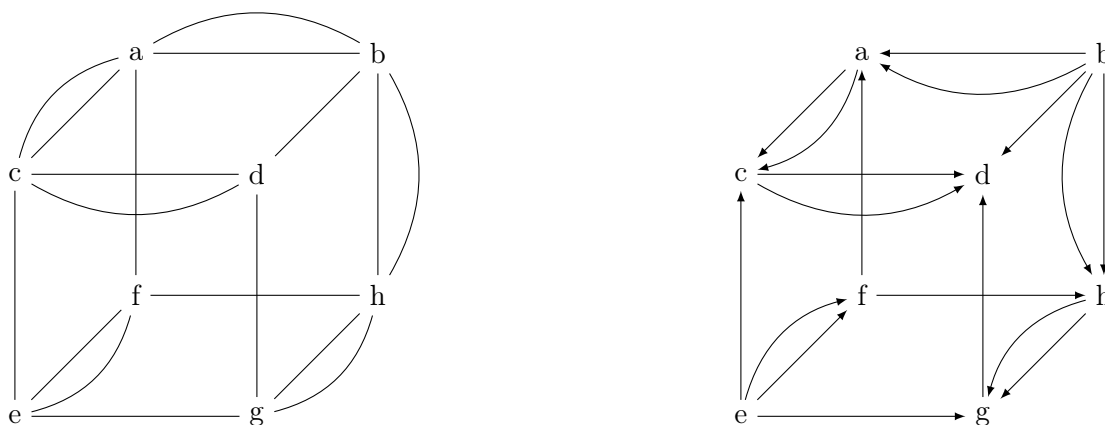


Figure 4.1: A multigraph G and an acyclic orientation of G

In [93] an algorithm for enumerating all acyclic orientations with delay $O(n^3)$ but linear amortized time $O(n)$ was given. Later, the delay was reduced to $O(mn)$ with an increase in amortized time to $O(m + n)$ [7]. Another improvement was obtained in [20], where an algorithm of delay $O(m)$ and the resulting amortized time $O(m)$, is given. Many more types of orientations have been studied with respect to their enumeration complexity, see [19] for an overview. A concept dual to acyclic orientations are *strongly connected* orientations i.e., those that for any two vertices $u, v \in V$ have a directed path from u to v .

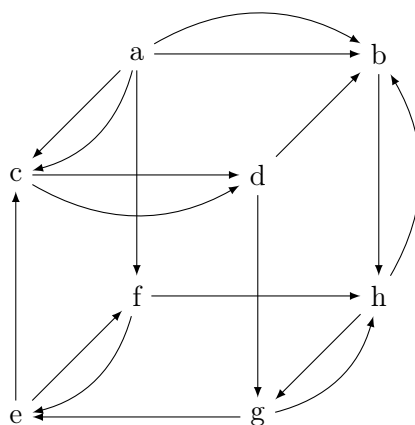


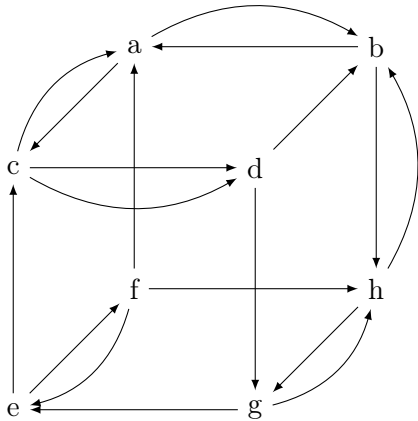
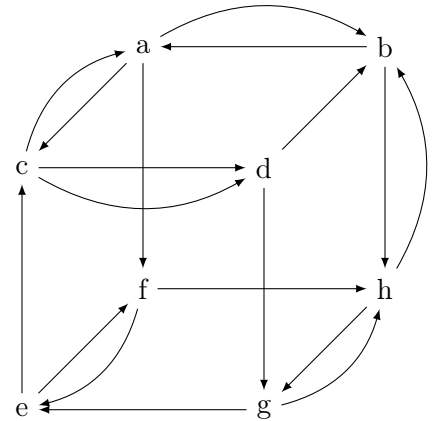
Figure 4.2: An orientation that is neither acyclic nor strongly connected

Figure 4.2 is an example of orientation that contains a cycle (c, d, g, e) but is not strongly connected: there is no way to go from vertex b to vertex a . For an example of strongly connected orientation, see Figure 4.3. In [21] an enumeration algorithm for strongly connected orientations with delay $O(m)$ is given.

In this chapter we study efficient algorithms for enumerating orientations that are a generalization of strongly connected orientations, namely *k -arc-connected orientations*.

Definition 49. An orientation $D = (V, A)$ of $G = (V, E)$ is k -arc-connected if for all $A' \subseteq A$ with $|A'| < k$, the digraph $D \setminus A'$ is strongly connected.

In other words, an orientation of a graph is k -arc-connected if it is strongly connected and if the removal of at least k arcs is needed to destroy strong connectivity.

Figure 4.3: 1-arc-connected orientation of G Figure 4.4: 2-arc-connected orientation of G

Figures 4.3 and 4.4 are orientations of the undirected graph G of the left part of Figure 4.1. Both orientations are strongly connected but, unlike in Figure 4.3, the orientation of Figure 4.4 is 2-arc-connected. Indeed, observe that despite the fact that the only difference between the two orientations is the direction of edge (a, f) , removing from graph 4.3 arc (g, e) gives rise to a graph that is no longer strongly connected (it is for example no more possible to go from vertex a to vertex e).

In this chapter, we will be interested in the following enumeration problem:

ENUM_ k -ARC-CONNECTED-ORIENTATIONS

Input : a graph $G = (V, E)$

Output : the set of all k -arc-connected orientations of G

In the following, when there is no ambiguity, we will sometimes abbreviate k -arc-connected by saying k -connected. A major contribution in the field of graph orientations under connectivity constraints is due to Robbins [87]. He characterized the graphs that admit a strongly connected orientation. Let us first define the notion of k -edge-connectivity.

Definition 50. An undirected graph $G = (V, E)$ is k -edge-connected if for all $E' \subseteq E$ with $|E'| < k$, the graph $G \setminus E'$ is connected.

In other words, an undirected graph is k -edge-connected if it is connected and if the removal of at least k edges is needed to destroy the connectivity.

Theorem 51 (Robbins [87]). A graph G admits a strongly connected orientation if and only if G is 2-edge-connected.

A generalization of this theorem is given for k -arc-connected orientations by Nash-Williams. His theorem is at the core of most of the results used in the chapter to come.

Theorem 52 (Nash-William weak orientation theorem [83]). *A graph $G = (V, E)$ admits a k -arc-connected orientation D if and only if G is $2k$ -edge-connected.*

Observe that the necessity of Theorem 52 above is trivial: suppose that $G = (V, E)$ admits a k -arc-connected orientation D . If $\deg(X)$ denotes the number of incoming and outgoing edges of $X \subseteq V$ and $\delta_D^+(X)$ denotes the number of outgoing arcs of X , then we have $\deg(X) = \delta_D^+(X) + \delta_D^-(X) \geq k + k = 2k$. Notice that by Definition 50, an undirected graph is $2k$ -edge-connected if and only if the degree of each non trivial vertex set is at least $2k$. This completes the necessity of Theorem 52. The difficulty of Theorem 52 is in the sufficiency. An independent proof of Lovász [75] answers this sufficiency condition; it relies on a decomposition of $2k$ -edge-connected graphs. More precisely he shows that a graph G is $2k$ -edge-connected if and only if G can be built starting with a pair of vertices connected by $2k$ edges with two operations: add an edge or *pinch* a set of k edges, i.e., add a new vertex s to V and replace each edge $(u_i, v_i) \in E$, $i \in \{1, \dots, k\}$ by two new edges (u_i, s) and (v_i, s) . This result yields an easy algorithm to produce a k -connected orientation of G (if there is one) with running time $O(n^6)$. The idea is to start by reducing the considered graph G to two vertices connected by $2k$ undirected edges. Then we rebuild the initial graph G by orienting the newly created edges. We orient k edges backwards and k edges forwards. When pinching arc (u_i, v_i) with vertex s , we keep the orientation property by orienting (u_i, s) and (s, v_i) . And when adding an edge without pinching, then we orient it arbitrarily. This construction starting from two vertices connected by $2k$ edges, is the reverse operation of a *splitting-off*. Other improvements of Lovász's algorithm have been developed in order to obtain a better running time, see [44, 74].

Important contributions to the theory of k -connected orientations come from Frank. In particular, he showed that any two k -connected orientations of G can be transformed into each other by reversing directed paths and directed cycles [40]. Disassembling this result lies at the core of the algorithms presented in this chapter. Another important contribution of Frank is the integration of k -connected orientations into the theory of submodular flows, see [39]. In particular, finding a submodular flow (and hence a k -connected orientation) can be done in polynomial time. There is a considerable amount of literature proposing different algorithmic solutions for this task on simple graphs, multigraphs, and mixed graphs, see e.g. [39, 43, 45, 62].

Using a minimum-cost k -connected orientation algorithm (for example [45, 62]) and following ideas of [5] one can design an algorithm that decides in $O(n^3k^3 + kn^2m)$ if a given mixed graph $G = (V, E \cup A)$, where some edges are oriented and others are not, can be extended to an orientation which is k -connected. This yields an enumeration algorithm for k -connected orientations with delay $O(m(n^3k^3 + kn^2m))$. The idea is to simply start with $G = (V, E)$ as the root vertex of the enumeration tree and at each node create two children by picking an edge, fixing it to be oriented in one way or the other, and verifying if a k -connected extension exists. This approach is an instance of constructing an enumeration algorithm out of EXTSOL. Subsections 4.3.1 and 4.3.2 state two ways of solving the extension problem - which consists in verifying if a k -connected extension exists - in polynomial time. Since this tree has depth in $O(m)$ the previously stated running time follows from [45, 62]. Note that any other algorithm finding a k -connected extension of a partial orientation could be used in this approach (for example algorithms from [39, 43, 45]). In particular, one can use submodular-flow feasibility. We outline both techniques in Section 4.3.

The main contribution of the present chapter is an alternative approach to solve the enumeration problem of k -connected orientations. We consider two enumeration problems of independent interest, whose combination specializes to the enumeration of k -connected orientations. The first one consists in enumerating all orientations of G with a prescribed outdegree sequence also known as α -orientations. Given $\alpha : V \rightarrow \mathbb{N}$, an orientation D of G is an α -orientation if $\delta_D^+(v) = \alpha(v)$ for all $v \in V$, where $\delta_D^+(v)$

denotes the outdegree of v . The second problem consists in the generation of all outdegree sequences for which there exists at least one k -connected orientation. For each of these problems, we propose and compare two solving methods. The main advantages of this approach are the simplicity of the final algorithm (while algorithms for submodular flows tend to be rather intricate) and that the two parts of this algorithm enumerate objects of independent interest. The idea basically comes from Frank's above mentioned result [40] that reversals of directed cycles and directed paths are enough to enumerate all k -connected orientations.

The α -orientations are of current interest with respect to computational properties (see e.g. [1]) and model many combinatorial objects such as domino and lozenge tilings of a plane region [86, 96], spanning trees of a planar graph [48], perfect matchings (and d -factors) of a bipartite graph [34, 73, 85], Schnyder woods of a planar triangulation [15], Eulerian orientations of a graph [34], and contact representations of planar graphs with homothetic triangles, rectangles, and k -gons [35, 37, 56]. We will present two algorithms for the enumeration of all α -orientations for some fixed α . The first is due to a reduction to the problem of enumerating the vertices of a polyhedron. Indeed, we will present two methods for enumerating the integer points of a box-integer polytope, that specializes to this problem, see Subsection 4.4.1. The second algorithm for generating the α -orientations is a flashlight backtrack search using BFS and is developed in Subsection 4.4.2. The second algorithm turns out to be the more efficient having a delay of $O(m^2)$. Furthermore, while the first approach relies on LP-solvers, the second approach is elementary and can easily be implemented from scratch. Note that the set of α -orientations of a planar graph can be endowed with a natural distributive lattice structure [34] and therefore, in this case the enumeration can be done in linear amortized time [58]. It is a famous question (also open in this setting of planar graphs) whether the enumeration of the elements of a distributive lattice can be done in constant amortized time [46].

Concerning the second problem that consists in enumerating, for a given graph, all outdegree sequences that are attained by k -connected orientations, we propose also two different ways of solving it. The first one is an illustration of the reverse search technique and the second more efficient one, is a flashlight backtrack search and has a delay of $O(m^2kn)$.

Finally, we combine the two most efficient algorithms for each enumeration problem, leading to an enumeration algorithm for k -connected orientations of delay $O(m^2kn)$ and amortized time $O(m^2)$. While the delay is similar to the algorithm obtained from extending k -connected orientations, no such algorithm is likely to yield amortized time $O(m^2)$.

Section 4.2 is devoted to some general preliminaries and to the study of Frank's theorem on the relation between k -connected orientations, which is central for the construction of our algorithms. Section 4.3 states a first flashlight backtrack search enumeration algorithm for generating all k -connected orientations based on algorithms finding a k -connected extension of a partial orientation. We present two such algorithms: one based on minimum-cost k -connected orientations and one on submodular flow feasibility. In Section 4.4 we present two ways of solving the α -orientations enumeration problem. In particular, we give an enumeration algorithm for the integer points of a box-integer polytope. In Section 4.5 we present two ways of solving the outdegree sequences enumeration problem. We state the final algorithm for enumerating all k -connected orientations as well as the analysis of the amortized time in Section 4.6. We conclude this chapter with further remarks in Section 4.7.

4.2 Preliminaries

After some recallings on orientations, we present the theorem by Frank which lies at the core of our algorithms.

4.2.1 Notations and basics of orientations

In this section we introduce some graph basics. All graphs we consider in this paper are multigraphs and consequently their orientations also may have multiple parallel or anti-parallel arcs. We will also consider *mixed (multi)graphs*. These are of the form $G = (V, E \cup A)$, where E is a (multi)set of undirected edges and A is a (multi)set of directed arcs. Analogously to undirected graphs, an orientation of a mixed graph consists in fixing a direction for each of its undirected edges.

Let $G = (V, A)$ be a directed graph. For any two vertices u and v , we will denote by P_{uv} a directed path from u to v . For D an orientation of G and a proper subset $X \subseteq V$, we set $A_D[X, V \setminus X]$ for the set of arcs of D from X to $V \setminus X$. Let $\delta_D^+(X) = |A_D[X, V \setminus X]|$ be the outdegree of X in D and $\delta_D^-(X) = |A_D[V \setminus X, X]|$ be the indegree of X in D . Assuming that $V(G) = \{x_1, \dots, x_n\}$, we set $\delta_D^+ = (\delta_D^+(x_1), \dots, \delta_D^+(x_n))$, the outdegree vector associated to D . For two vectors α and α' in \mathbb{N}^n , we define the notion of distance between them via the ℓ_1 -metric, i.e., $d(\alpha, \alpha') = \sum_{i=1}^n |\alpha_i - \alpha'_i|$. The digraph obtained from $D = (V, A)$ by reversing a set of arcs $B \subseteq A$ is denoted by D^B and the reversed arc set of B is denoted $B^- = \{(u, v) \mid (v, u) \in B\}$. In particular, the inverse of an arc a is denoted a^- . If $A = B$ we might write D^- instead of D^B .

A useful (and crucial) property of the outdegree function is the following (see Figure 4.5 for an illustration):

Remark 3. Let $G = (V, E)$ and $X, Y \subseteq V$ such that $X \cap Y \neq \emptyset$ and $X \cup Y \neq V$. We have the following inequality:

$$\delta_D^+(X) + \delta_D^+(Y) \geq \delta_D^+(X \cap Y) + \delta_D^+(X \cup Y)$$

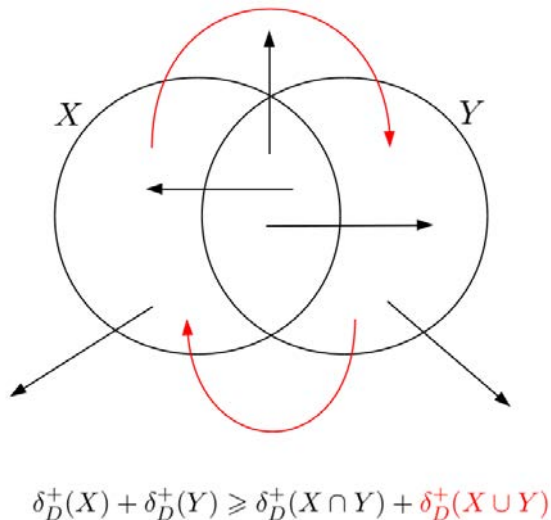


Figure 4.5: Illustration of Remark 3

In the Figure 4.5 above, the possible types of arcs of $\delta_D^+(X) \cup \delta_D^+(Y)$ are represented by the set of black and red arcs. Observe that there might exist some arcs that belong to $\delta_D^+(X) \cup \delta_D^+(Y)$ and not to $\delta_D^+(X \cup Y)$. These arcs are represented in red in the figure.

With respect to arc-connectivity in digraphs there is a generalization of Menger's Theorem [82], that we will make frequent use of. Denoting by $\lambda(u, v)$ the maximum number of arc-disjoint directed paths from u to v we can formulate it as follows:

Theorem 53 (Local Menger Theorem for digraphs). *Let $D = (V, A)$ be a digraph and $u, v \in V$. We have $\lambda(u, v) = \min\{\delta_D^+(X) \mid u \in X \not\ni v\}$.*

Note that by the definition of k -connectivity D is k -connected if and only if $\delta_D^+(X) \geq k$ for every non-empty proper subset $X \subset V$. Thus, as a consequence of Theorem 53, D is k -connected if and only $\lambda(u, v) \geq k$ for all $u, v \in V$. Hence, for a directed graph D the following statements are equivalent:

- D is k -arc-connected;
- removing less than k arcs of D results in a strongly connected graph;
- the outdegree of each non trivial vertex set is at least k ;
- for all pairs (u, v) of vertices, there exists k directed and arc-disjoint paths from u to v .

4.2.2 Frank's theorem on k -arc-connected orientations

In [40], Frank states a theorem, the proof of which gives us information on the link between two k -arc-connected orientations D and D' of some graph G and on their outdegrees.

Theorem 54 (Frank [40]). *If D and D' are two k -arc-connected orientations of some graph G , then there is a sequence $D = D_0, D_1, \dots, D_p = D'$ of k -arc-connected orientations of G such that each D_i ($i = 1, 2, \dots, p$) arises from D_{i-1} by reversing one directed cycle or path.*

We will here reformulate Frank's proof of the above theorem and extend his result by putting forward the main arguments needed to solve ENUM_ k -ARC-CONNECTED-ORIENTATIONS.

If D and D' are two k -arc-connected orientations of some graph G , in order to transform D into D' by reversals of directed paths and directed cycles, there are two cases to consider:

1. $\delta_D^+ = \delta_{D'}^+$, which is explained by Lemma 55,
2. $\delta_D^+ \neq \delta_{D'}^+$, which is explained by Lemma 58.

Lemma 55. *Let G be some graph and D and D' be two orientations of G . We have $\delta_D^+ = \delta_{D'}^+$, if and only if orientation D' can be obtained from D by reversing a set of arc-disjoint directed cycles.*

Proof. Reversing the direction of a directed cycle does not change the outdegree function; hence, if D' is obtained from D by reversing a set of arc-disjoint directed cycles, then $\delta_D^+ = \delta_{D'}^+$.

Now, suppose that orientations $D = (V, A)$ and $D' = (V, A')$ are such that $\delta_D^+ = \delta_{D'}^+$, and let us show that D' can be obtained from D by reversing some arc-disjoint directed cycles. Consider the arcs of D whose direction differs in D' , i.e., $A \setminus A'$. Observe that they form a directed subgraph $D \setminus D'$ such that the indegree and the outdegree coincide at each node; otherwise we would have $\delta_D^+ \neq \delta_{D'}^+$. It is well-known that a connected digraph with $\delta^+(v) = \delta^-(v)$ is Eulerian, i.e., there is a directed cycle using every arc exactly once. Furthermore, it is well-known that Eulerian digraphs are arc-disjoint unions of directed cycles. Thus $D \setminus D'$ is an arc-disjoint union of directed cycles, which concludes the proof. \square

Now, let us look at the case where $\delta_D^+ \neq \delta_{D'}^+$.

Claim 56. *Let D be an orientation of a graph G and $X \subseteq V(G)$. If D' is obtained from D by reversing a path from a vertex u to a vertex v , then we have*

1. $\delta_{D'}^+(X) = \delta_D^+(X)$ if $u, v \in X$ or $u, v \notin X$
2. $\delta_{D'}^+(X) = \delta_D^+(X) + 1$ if $u \notin X$ and $v \in X$
3. $\delta_{D'}^+(X) = \delta_D^+(X) - 1$ if $u \in X$ and $v \notin X$

Proof. We consider a path P_{uv} and $A_D[X, V \setminus X]$, i.e., the outgoing arcs of X in orientation D . First, consider the case (3) of Claim 56, where $u \in X$ and $v \notin X$. Say $|P_{uv} \cap A_D[X, V \setminus X]| = p$. Because $u \in X$ and $v \notin X$, $|P_{uv} \cap A_D[V \setminus X, X]| = p - 1$ and thus after the reversal of P_{uv} , there are $p - 1$ outgoing arcs of X .

This case is illustrated by Figure 4.6 below, where the red arcs are the arcs considered in Claim 56, namely the outgoing arcs of X . The blue arcs are the incoming arcs of X and the black dotted arcs represent possible paths inside X or \bar{X} .

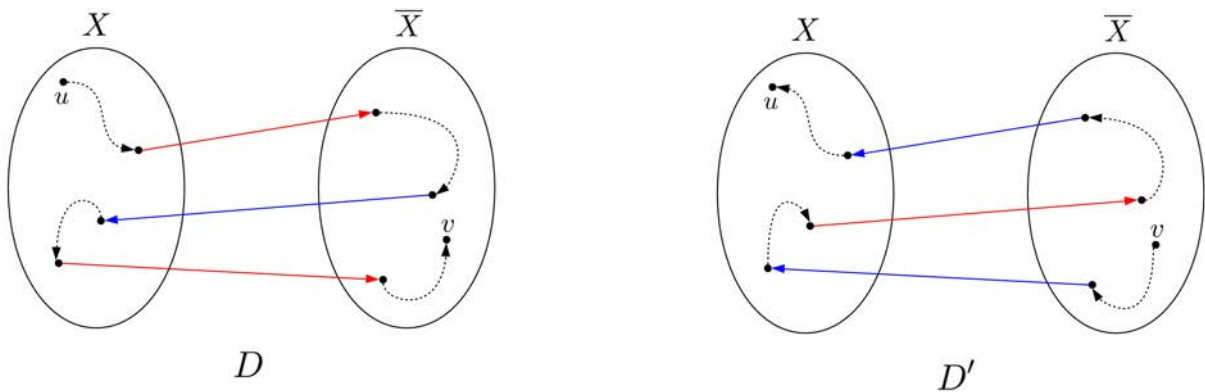


Figure 4.6: Claim 56 Case (3)

The case (2) of Claim 56, where $u \notin X$ and $v \in X$ is similar to case (3) we just explained. Figure 4.7 below is an illustration.

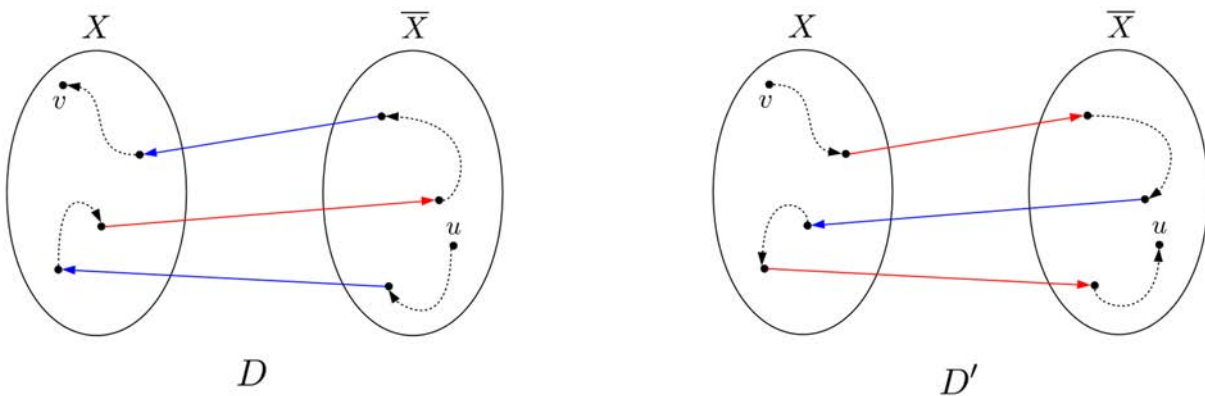


Figure 4.7: Claim 56 Case (2)

Now, if $u, v \in X$ or $u, v \notin X$, then $P_{uv} \cap A_D[X, V \setminus X] = \emptyset$ and thus the number of outgoing arcs of X does not change after the reversal of P_{uv} , see Figure 4.8 below.

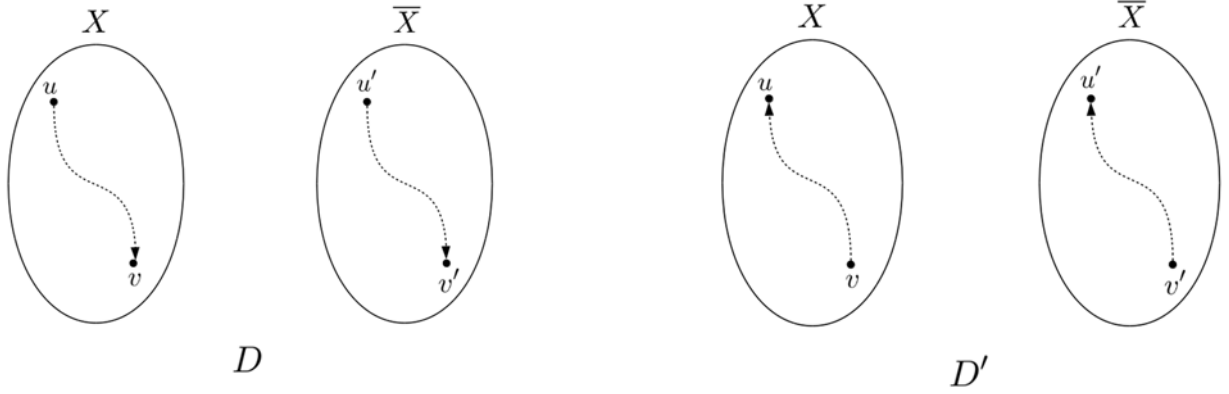


Figure 4.8: Claim 56 Case (1)

□

Claim 56 implies the following lemma:

Lemma 57. *Let D and D' be two orientations of some graph G such that D' can be obtained from D by reversing one directed path. Then*

- (1) *there exists a vertex u such that $\delta_{D'}^+(u) = \delta_D^+(u) - 1$;*
- (2) *there exists a vertex v such that $\delta_{D'}^+(v) = \delta_D^+(v) + 1$;*
- (3) *for all other vertices $w \neq u, v$ we have $\delta_{D'}^+(w) = \delta_D^+(w)$.*

and hence $d(\delta_D^+, \delta_{D'}^+) = 2$.

Notice that the converse of Lemma 57 does not hold. In fact, if D and D' are two (k -arc-connected) orientations such that (1), (2) and (3) are satisfied, then D' can be obtained from D by reversing a directed path and some arc-disjoint directed cycles.

More generally, we have Lemma 58 presented below which is a generalization of Theorem 54.

In a k -arc-connected orientation D we call a directed path P *flippable* if D^P is k -arc-connected. We will call a pair (u, v) *flippable* if there exists a flippable path P_{uv} from u to v .

Lemma 58. *Let G be a graph and D, D' be two k -arc-connected orientations of G such that $\delta_D^+ \neq \delta_{D'}^+$. Then, the following holds:*

1. *for each v such that $\delta_D^+(v) < \delta_{D'}^+(v)$, there exists a vertex u such that $\delta_D^+(u) > \delta_{D'}^+(u)$ and (u, v) is flippable in D ;*
2. *for each v such that $\delta_D^+(v) > \delta_{D'}^+(v)$, there exists a vertex u such that $\delta_D^+(u) < \delta_{D'}^+(u)$ and (v, u) is flippable in D .*

In both cases, orientation D'' obtained after the reversal of P_{uv} or P_{vu} is such that $d(\delta_{D''}^+, \delta_{D'}^+) = d(\delta_D^+, \delta_{D'}^+) - 2$.

Proof. We start by recalling the proof of Frank of point (1), and then we develop two ways of proving point (2).

Proof of point (1): Let v be such that $\delta_D^+(v) < \delta_{D'}^+(v)$. We will show (4.1) below, from which we will be able to deduce the existence of a flippable path from u to v in D .

$$\begin{aligned} &\text{There exists a vertex } u \text{ such that } \delta_D^+(u) > \delta_{D'}^+(u) \\ &\text{and } \delta_D^+(X) > k \text{ whenever } u \in X \text{ and } v \notin X \end{aligned} \quad (4.1)$$

Notice that because we have v such that $\delta_D^+(v) < \delta_{D'}^+(v)$ and because $\sum_{w \in V} \delta_D^+(w) = |E| = \sum_{w \in V} \delta_{D'}^+(w)$, there exists a vertex u such that $\delta_D^+(u) > \delta_{D'}^+(u)$. By contradiction suppose the following.

$$\begin{aligned} &\text{For each vertex } u \text{ such that } \delta_D^+(u) > \delta_{D'}^+(u) \\ &\text{there exists a set } X \text{ such that } u \in X, v \notin X \text{ and } \delta_D^+(X) = k \end{aligned}$$

Consider the maximal sets V_i for which $\delta_D^+(V_i) = k$ and $v \notin V_i$ for $i \in \{1, \dots, t\}$. Set $V_0 = V - \bigcup V_i$. First of all let us explain why for all $1 \leq i, j \leq t$ such that $i \neq j$, the sets V_i and V_j are disjoint. By contradiction suppose that $V_i \cap V_j \neq \emptyset$. We know that because $v \notin V_i \cup V_j$, we have $V_i \cup V_j \neq V$. Thus, by Remark 3 we would have

$$k + k = \delta_D^+(V_i) + \delta_D^+(V_j) \geq \delta_D^+(V_i \cup V_j) + \delta_D^+(V_i \cap V_j) \geq k + k \quad (4.2)$$

By maximality $V_i \not\subseteq V_j$ and $V_j \not\subseteq V_i$ and 4.2 implies $\delta_D^+(V_i \cup V_j) = k$, a contradiction with the maximality of V_i and V_j .

Now, in both D and D' let us count the number of edges having at most one end vertex in some V_i . Let us denote by c (resp. c') the number of edges in D (resp in D') having exactly one extremity in some V_i plus the number of edges having no extremity in $\bigcup V_i$ at all. The orientations D and D' differ only on the directions of their arcs and so we should have $c = c'$. Let us show that this is not the case. Observe that

$$\begin{aligned} c &= \sum_{i=1}^t \delta_D^+(V_i) + \sum_{z \in V_0} \delta_D^+(z) & c' &= \sum_{i=1}^t \delta_{D'}^+(V_i) + \sum_{z \in V_0} \delta_{D'}^+(z) \\ &= kt + \sum_{z \in V_0} \delta_D^+(z) & &= \delta_{D'}^+(V_i)t + \sum_{z \in V_0} \delta_{D'}^+(z) \end{aligned}$$

Recall that if w is such that $\delta_D^+(w) > \delta_{D'}^+(w)$, then $w \in \bigcup V_i$, thus for all $z \in V_0$, we have $\delta_D^+(z) \leq \delta_{D'}^+(z)$. Furthermore, recall that vertex v is such that $\delta_D^+(v) < \delta_{D'}^+(v)$. Hence we have

$$\sum_{z \in V_0} \delta_D^+(z) < \sum_{z \in V_0} \delta_{D'}^+(z)$$

Finally, D' being k -arc-connected, $k \leq \delta_{D'}^+(V_i)$ and so we can conclude:

$$c = kt + \sum_{z \in V_0} \delta_D^+(z) < kt + \sum_{z \in V_0} \delta_{D'}^+(z) \leq \delta_{D'}^+(V_i)t + \sum_{z \in V_0} \delta_{D'}^+(z) = c'$$

This completes the proof of (4.1).

Now, let D'' be the orientation obtained from D by reversing a path from u to v . Because D is k -arc-connected and by Claim 56, we can conclude that D'' is k -arc-connected as well:

Case 1: $u, v \notin X$ or $u, v \in X$, then $\delta_{D''}^+(X) = \delta_D^+(X) \geq k$;

Case 2: $v \in X$ and $u \notin X$, then $\delta_{D''}^+(X) > \delta_D^+(X) \geq k$;

Case 3: $u \in X$ and $v \notin X$, then $\delta_{D''}^+(X) = \delta_D^+(X) - 1$, but by (4.1) $\delta_D^+(X) > k$, thus $\delta_{D''}^+(X) \geq k$.

Finally, notice that by Lemma 57 we have $d(D'', D') = d(D, D') - 2$.

Now, let us prove point (2): We choose here to present two ways of proving this result. The first one is an attempt to stay as close as possible to Frank's reasoning of point (1) presented above. It allows us a better understanding of what hides behind Lemma 58. The second one, faster, is a beautiful and direct way of proving (1) using (2).

Point (2), proof 1: Let v be such that $\delta_D^+(v) > \delta_{D'}^+(v)$. Similarly to point (1) we will show (4.3) below, from which we will be able to deduce the existence of a flippable path from v to u in D .

$$\begin{aligned} &\text{There exists a vertex } u \text{ such that } \delta_D^+(u) < \delta_{D'}^+(u) \\ &\text{and } \delta_D^+(X) > k \text{ whenever } v \in X \text{ and } u \notin X \end{aligned} \quad (4.3)$$

For the same reasons than in point (1), there exists a vertex u with $\delta_D^+(u) < \delta_{D'}^+(u)$. By contradiction suppose the following.

$$\begin{aligned} &\text{For each vertex } u \text{ such that } \delta_D^+(u) < \delta_{D'}^+(u) \\ &\text{there exists a set } X \text{ such that } v \in X, u \notin X \text{ and } \delta_D^+(X) = k \end{aligned} \quad (4.4)$$

Consider the maximal sets X_i for which $\delta_D^+(X_i) = k$ and $v \in X_i$. First of all let us explain that for all i, j with $i \neq j$, we have $X_i \cup X_j = V$. Suppose by contradiction that $X_i \cup X_j \neq V$. We know that because $v \in X_i$ and $v \in X_j$, we have $X_i \cap X_j \neq \emptyset$. Thus, by Remark 3 we would have

$$k + k = \delta_D^+(X_i) + \delta_D^+(X_j) \geq \delta_D^+(X_i \cup X_j) + \delta_D^+(X_i \cap X_j)$$

As for point (1), this would imply that $\delta_D^+(X_i \cup X_j) = k$, a contradiction with the maximality of X_i and X_j .

Now, consider $V_i = \overline{X_i}$. For all i, j with $i \neq j$ the sets V_i, V_j are disjoint: $\overline{X_i} \cap \overline{X_j} = \overline{X_i \cup X_j} = \overline{V} = \emptyset$. It is now possible to unfold the rest of the proof in the same way it is done for point (1) with, this time, $V_i = \overline{X_i}$ and $V_0 = \bigcup_i V_i = \bigcap_i \overline{V_i} = \bigcap_i \overline{X_i} = \bigcap_i X_i$. The contradiction lies again in the number of edges with one extremity in some V_i plus the number of edges having no extremity in some V_i at all, which does not coincide in D and D' .

Point (2), proof 2: Observe that an equivalent statement of the lemma is that for every vertex v such that $\delta_D^-(v) < \delta_{D'}^-(v)$, there exists a vertex u such that $\delta_D^-(u) > \delta_{D'}^-(u)$ and (v, u) is flippable in D . By fully reorienting all the arcs of D and D' , we obtain graphs D^- and D'^- and get a further equivalent statement: for every vertex v such that $\delta_{D^-}^+(v) < \delta_{D'^-}^+(v)$, there exists a vertex u such that $\delta_{D^-}^+(u) > \delta_{D'^-}^+(u)$ and (u, v) is flippable in D^- . This is precisely the statement of Lemma 58 point (1), so we are done. □

The proof of Theorem 54 boils down to Lemmas 58 and 55. If D and D' are two k -arc-connected orientations of some graph G , then repeating the procedure of Lemma 58, either Lemma 58 or Lemma 55 will occur.

4.3 Enumeration of k -connected orientations via submodular flows

A natural way to enumerate k -connected orientations of a given graph is to try both possible orientations of a given undirected edge and check whether the obtained mixed graph can be extended to a k -connected orientation. If yes, then we recurse. Such a method is doable provided that we can answer in efficient time to the question "*can a partial orientation be extended to a k -connected orientation ?*". Note that this approach is an instance of constructing an enumeration algorithm out of EXTSOL, see Section 2.1.

Algorithm 14: Enumeration of k -connected orientations via submodular flows

```

Input: a graph  $G = (V, E)$  and an integer  $k$ 
Output: the  $k$ -connected orientations of  $G$ 

1 begin
2   | Fix any linear ordering on  $E$ ;
3   | Enumerate( $G = (V, E, \emptyset)$ );
4 end

5 Function Enumerate( $G = (V, E, F)$ ):
6   | if  $E \neq \emptyset$  then
7     |   Take the smallest  $a = (u, v) \in E$ ;
8     |   if  $G' = (V, E \setminus \{a\}, F \cup \{(u, v)\})$  admits a  $k$ -connected orientation then
9     |     | Enumerate( $G'$ );
10    |   if  $G' = (V, E \setminus \{a\}, F \cup \{(v, u)\})$  admits a  $k$ -connected orientation then
11    |     | Enumerate( $G'$ );
12    | else
13    |   | Output  $G$ ;
14 end

```

Here we cite two methods to answer this question. The first one uses the idea of Frank [41] to reduce this problem to submodular flow feasibility. The second method is to reduce the problem to finding a minimum-cost k -connected orientation (see for example [5]). We define the latter as follows.

MINIMUM_COST-k-CONNECTED-ORIENTATION	
<i>Input :</i>	G an undirected graph, a cost for each possible edge direction of every edge
<i>Output :</i>	a minimum cost k -connected orientation

4.3.1 Partial orientation extension with submodular flow feasibility

Frank shows that the problem of finding a k -arc-connected orientation is a special case of the submodular flow problem [41] and Gabow proposes improvements of Frank's algorithm [43]. Let us introduce the notion of submodular flow.

Let V be a finite ground set. Sets $X, Y \subseteq V$ are *intersecting* if none of $X \cap Y$, $X \setminus Y$ and $Y \setminus X$ is empty. If, in addition, $X \cup Y \neq V$, then X, Y are *crossing*. A family \mathcal{F} of subsets of V is crossing if for all sets $X, Y \in \mathcal{F}$ that are crossing, we have that $X \cap Y$ and $X \cup Y$ belong to \mathcal{F} . A set function h is *submodular* on X, Y if $h(X) + h(Y) \geq h(X \cap Y) + h(X \cup Y)$. We may refer to a pair (h, \mathcal{F}) as a crossing submodular function if \mathcal{F} is a crossing family and h a function on \mathcal{F} , submodular on all crossing pairs of \mathcal{F} .

Remark 4. *The family $2^V - \{\emptyset, V\}$ is crossing. Moreover, the pair $(\delta_D^+, 2^V - \{\emptyset, V\})$ is a crossing submodular function (recall Remark 3).*

For a function $f \in \mathbb{R}^A$ and any subset $X \subseteq A$, we set $f(X)$ for $\sum\{f(e) \mid e \in X\}$. We define the notion of submodular flow problem as follows.

Definition 59. *Let $G = (V, A)$ be a directed graph with orientation D . Let (h, \mathcal{F}) be a crossing submodular function. Let f and g be functions such that $f \in (\mathbb{R} \cup \{-\infty\})^A$ and $g \in (\mathbb{R} \cup \{+\infty\})^A$ and let $d \in \mathbb{R}^A$ be a weight function on the arcs. A submodular flow problem consists in finding a solution $x \in \mathbb{R}^A$, which is called a submodular flow, solving the linear program defined below.*

$$\begin{cases} \min dx(A) \\ x(A_D[Y, V \setminus Y]) - x(A_D[V \setminus Y, Y]) \leq h(Y) \text{ for all } Y \in \mathcal{F} \\ f \leq x \leq g \end{cases} \quad (4.5)$$

where $dx(A)$ denotes $\sum\{d(e)x(e) \mid e \in A\}$.

Sometimes submodular flow problems are also called *Edmonds-Giles problem* and instead of minimization, maximization is considered. We will use the term *feasible solution* when speaking of a solution to the inequality system underlying the optimization problem and we will use the term *optimal solution* when speaking of a solution to the optimization problem. Such submodular flow problem is said to be a $0 - 1$ submodular flow problem if $d \geq 0$, h is integral and $f \equiv \mathbf{0}$ and $g \equiv \mathbf{1}$, i.e., f is the constant 0-function and g is the constant 1-function.

Proposition 60 ([41]). *Let $G = (V, A \cup E)$ be a mixed graph. The problem of finding a k -arc-connected orientation of G respecting A reduces to the problem of finding a feasible solution of a submodular flow problem.*

Let us recall the proof of this proposition.

Proof. We will show that there is a bijection between the k -arc-connected orientations of a mixed graph $G = (V, A \cup E)$ and the integer-valued solutions of a particular submodular flow problem. Let D_0 be a reference orientation of G obtained by preserving the orientations of A and orienting the edges of E arbitrarily and let $D'_0 = D_0 \setminus A$. Let x be a function such that $x \in \mathbb{R}^E$, and consider the following system of inequalities.

$$\begin{cases} x(A_{D'_0}[X, V \setminus X]) - x(A_{D'_0}[V \setminus X, X]) \leq \delta_{D'_0}^+(X) - k & \forall X \in 2^V - \{\emptyset, V\} \\ 0 \leq x \leq 1 \end{cases} \quad (4.6)$$

The system of inequalities (4.6) can be seen as a 0–1 submodular flow problem if we consider the associated optimization problem with function $d \in \mathbb{R}^E$ such that $d = 0$ (hence, each feasible solution is optimal). In fact, by Remark 4 we know that the pair $(\delta_{D'_0}^+(X) - k, 2^V - \{\emptyset, V\})$ is a crossing submodular function.

Now let us show that there is a one-to-one correspondence between the integer-valued solutions of (4.6) and the k -arc-connected orientations of G . The proof boils down to the following observation, which is illustrated through Figure 4.9.

Observation 1. *Let $G = (V, A \cup E)$ be a mixed graph and let D and D' be two orientations of G which coincide on A . Let function $x \in \mathbb{R}^{E \cup A}$ be such that $x(e) = 0$ if arc e has the same direction in D and D' and $x(e) = 1$ if it differs. Then,*

$$\delta_{D'}^+(X) = \delta_D^+(X) - x(A_D[X, V \setminus X]) + x(A_D[V \setminus X, X]).$$

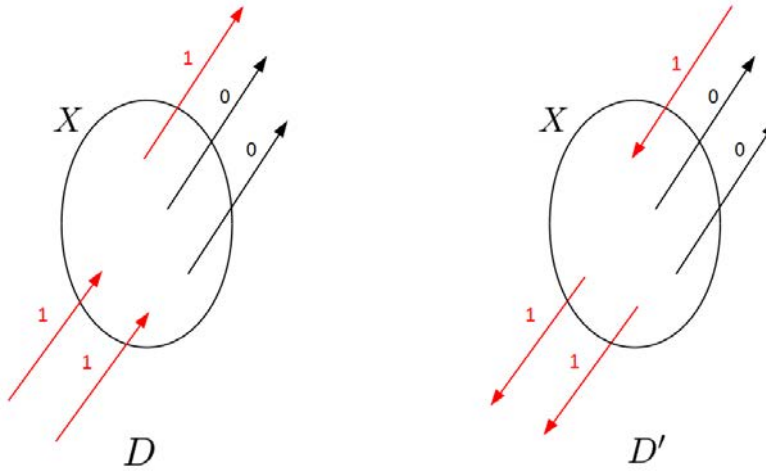


Figure 4.9: $\delta_{D'}^+(X) = \delta_D^+(X) - x(A_D[X, V \setminus X]) + x(A_D[V \setminus X, X]) = 3 - 1 + 2 = 4$

Now, let us prove the bijection. If x is an integer-valued solution of (4.6), we can construct the k -arc-connected orientation D as follows. First, we consider the extension \tilde{x} of x on $A \cup E$: we set $\tilde{x}(e) = 0$ for all $e \in A$. We then construct D from D_0 : if $\tilde{x}(e) = 0$, then we keep arc e and if $\tilde{x}(e) = 1$, then we reverse the direction of arc e . Notice that $\tilde{x}(A_{D_0}[X, V \setminus X]) = x(A_{D'_0}[X, V \setminus X])$ and $\tilde{x}(A_{D_0}[V \setminus X, X]) = x(A_{D'_0}[V \setminus X, X])$. Orientation D is k -arc-connected. In fact, by Observation 1 and by (4.6), we have for every proper subset $X \subseteq V$:

$$\begin{aligned} \delta_D^+(X) &= \delta_{D_0}^+(X) - \tilde{x}(A_{D_0}[X, V \setminus X]) + \tilde{x}(A_{D_0}[V \setminus X, X]) \\ &= \delta_{D_0}^+(X) - x(A_{D'_0}[X, V \setminus X]) + x(A_{D'_0}[V \setminus X, X]) \geq k \end{aligned} \quad (4.7)$$

Now, if D is a k -arc-connected orientation of graph G , we construct the integral solution $x \in \mathbb{R}^E$ of (4.6) as follows. First we construct $y \in \mathbb{R}^{E \cup A}$: we fix $y(e) = 0$ if arc e has the same direction in D_0 and D and $y(e) = 1$ if it differs. We define $x = y|_E$, i.e., x is the restriction of y to E . Because of Observation 1 and because D is k -arc-connected, we obtain formula (4.7), from which we can conclude that x is an integer-valued solution of (4.6). □

We have seen with Proposition 60 that the question whether a mixed graph admits a k -arc-connected orientation reduces to the question whether the submodular flow problem (4.6) admits an integral-valued solution. It turns out that (4.6) is a 0 – 1 submodular flow problem; for such submodular flow problems we can take advantage of the following result by Edmonds and Giles.

Theorem 61 ([39]). *A 0 – 1 submodular flow problem has an integral optimal solution provided that it has a feasible solution at all.*

Thus, we know that (4.6) has an integral solution if it is feasible. About the feasibility problem of a submodular flow, we know by Frank that it is polynomial but it is hard to find the best complexity in the literature. In [43], an algorithm in $O(kn^2(\sqrt{kn} + k^2 \log(n/k)))$ is presented but it works only for simple directed graphs. For multigraphs a complexity in $O(n^5 m (\log \frac{n^2}{m}))$ is given in [94]. Anyways, in the setting of mixed multigraphs we find a better analysis from the following approach.

4.3.2 Partial orientation extension using minimum-cost orientation

A second way of showing that a partial orientation can be extended to a k -connected orientation consists in using a minimum-cost k -connected orientation algorithm, see for example [5]. The idea is simply to unorient the whole mixed graph $G = (V, E \cup A)$, fix a cost 0 for the arcs of A and a cost 1 for the arcs of A^- and find a minimum-cost k -connected orientation of G . If at the end the cost is equal to 0, then there exists a k -connected extension for A , otherwise there is no extension. Some polynomial time minimum-cost k -connected orientation algorithms are given in [45, 62]. These algorithms use submodular flows.

Theorem 62 ([5, 62]). *Deciding whether a given mixed graph $G = (V, E \cup A)$ admits a k -connected orientation respecting A can be done in time $O(k^3 n^3 + kn^2 m)$.*

With Theorem 62 at hand, flashlight backtrack search Algorithm 14 enumerates all k -connected orientations with polynomial delay. The algorithm takes as input an undirected graph $G = (V, E)$. Clearly, all k -connected orientations are produced and since each node built by the algorithm gives rise to disjoint branches it does not repeat the same solution twice. The depth of the binary execution tree is m and at each node we check the orientability of a mixed graph which is solvable in $O(k^3 n^3 + kn^2 m)$ by Theorem 62. Thus, we obtain the following result:

Theorem 63. *Let G be a graph and $k \in \mathbb{N}$. Algorithm 14 enumerates all k -connected orientations of G with delay $O(m(k^3 n^3 + kn^2 m))$.*

The downside of Algorithm 14 and the main motivation for the rest of this chapter is that the complete understanding of the proof of Theorem 62 is rather intricate and the implementation of this step is not elementary.

In the following, we will divide ENUM_ k -ARC-CONNECTED-ORIENTATIONS into two subproblems of independent interest: ENUM_ α -ORIENTATIONS and ENUM_ k -CONNECTED-OUTDEGREE-SEQUENCES. The first one is the problem of enumerating all orientations respecting some fixed outdegree sequence (i.e. the outdegrees of all vertices of the graph). The second one consists in the enumeration of all outdegree

sequences for which there exists at least one k -connected orientation. For each of these subproblems we will analyse two different solving methods and finally pick the most efficient one. Together these two algorithms will give an alternative solution for the enumeration of k -connected orientations. Moreover, the resulting algorithm has a better delay and even better amortized running time.

4.4 Orientations with prescribed outdegree sequence

Let $G = (V, E)$ be a graph and $\alpha : V \rightarrow \mathbb{N}$. We say that an orientation D of G is an α -orientation if $\delta_D^+ \equiv \alpha$, i.e., $\delta_D^+(v) = \alpha(v)$ for all $v \in V$. We will denote by $\mathcal{O}_\alpha(G)$ the set of all α -orientations of G .

The following is the enumeration problem to be discussed in the present section.

ENUM_ α -ORIENTATIONS	
<i>Input</i> :	a graph $G = (V, E)$ and a function $\alpha : V \rightarrow \mathbb{N}$
<i>Output</i> :	the elements of $\mathcal{O}_\alpha(G)$

First of all, notice the following lemma, which will be useful in the sequel.

Lemma 64. *If D, D' are two orientations in $\mathcal{O}_\alpha(G)$, then we have $\lambda_D(u, v) = \lambda_{D'}(u, v)$.*

Proof. Recall that by Theorem 53, it is enough to prove that $\min\{\delta_D^+(X) \mid u \in X, v \notin X\} = \min\{\delta_{D'}^+(X) \mid u \in X, v \notin X\}$. Thus, let us show that $\delta_D^+(X) = \delta_{D'}^+(X)$. Consider $D, D' \in \mathcal{O}_\alpha(G)$ and let D be k -arc-connected, i.e., $\forall X \subseteq V$ we have $\delta_D^+(X) \geq k$. From Lemma 55, we know that because $\delta_D^+ = \delta_{D'}^+$, orientation D' can be obtained from D by reversing a set of disjoint directed cycles. Notice that reversing the direction of a directed cycle in D does not change the outdegree of the subsets of vertices of G . In fact, let C be a directed cycle of D that is reversed in D' and let $a \in D$ be an outgoing arc of some subset of vertices $X \subseteq V(G)$. If $a \notin C$, then a is still an arc that leaves X after the reversal of C . Now if $a \in C$, because C is a directed cycle, there exists another arc a' in C that enters X . After the reversal of C , a will enter X and a' will leave X . Hence, for all $X \subseteq V$ we have $\delta_D^+(X) = \delta_{D'}^+(X)$. \square

Theorem 65. *The problem ENUM_ α -ORIENTATIONS can be solved with a delay in $O(m^2)$. In particular, the problem belongs to DelayP.*

We will prove that ENUM_ α -ORIENTATIONS belongs to the complexity class DelayP using two different methods. The first one, stated in Section 4.4.1, is a reduction to the problem of enumerating the vertices of a polyhedron. The second one, stated in Section 4.4.2, is a simple flashlight backtrack search using Lemma 55. The second method turns out to be more efficient, and yield the desired complexity $O(m^2)$ of Theorem 65 above. Thus in the end it will be the one retained for the final algorithm enumerating all k -connected orientations.

4.4.1 Flashlight backtrack search via integer points of polytopes

We want to enumerate all orientations that respect some vector α , of some graph $G = (V, E)$. Our first method consists in reducing this problem to the problem of enumerating the vertices of a polytope.

A polyhedron P is the solution set to a system of q inequalities with d variables:

$$P = \{y \in \mathbb{R}^d \mid Ay \leq b\} \tag{4.8}$$

where A is a $q \times d$ matrix and b is a q -vector. A polyhedron P is convex, i.e., if $x, y \in P$, then the segment $[x, y] \in P$. From this definition follows the statement below.

Observation 2. If $P \subseteq \mathbb{R}^d$ is a polyhedron with $x, y \in P$ such that $x_i < y_i$, then for all $x_i \leq z_i \leq y_i$, there exists $g \in P$ such that $g_i = z_i$.

A *face* of a polyhedron P is a set of points of P that satisfies a valid inequality with equality. A *vertex* of a polyhedron P is a vector $y \in P$ that satisfies a linearly independent set of d inequalities as equalities. If P is bounded, then it is called a *polytope*. A polytope P is the convex hull of its vertices, i.e., the smallest convex set containing them [101]. We say that polytope P is *feasible* if $P \neq \emptyset$. If the vertices of a polytope are 0/1-vectors, then it is called a *0/1-polytope* [17]. A 0/1-polytope is included in $[0, 1]^d$. A polytope is *integer* if all its vertices have only integer coordinates. Furthermore, we call a polytope P *box-integer* if $P \cap \{x \mid l \leq x \leq u\}$ is an integer polytope for each choice of integer vectors l, u . The set $\{x \in \mathbb{R}^d \mid l \leq x \leq u\}$ is called a *box* and is denoted by $\mathbf{B}(l, u)$.

Lemma 66. A 0/1-polytope is box-integer. Furthermore, in a 0/1-polytope, the vertices correspond to the integer points.

Proof. First let us explain why a 0/1-polytope is box-integer. We want to show that the intersection of a 0/1-polytope P with a box $B(l, u)$ is an integer polytope. Without loss of generality we can state that a box $B(l, u)$ that has non-trivial intersection with P is of the form $l, u \in \{0, 1\}^d$ and $l_i = u_i$ for at least one $i \in \{1, \dots, d\}$ (otherwise the intersection is P). Thus, it suffices to show that 0/1-polytopes are preserved by intersection with hyperplanes of the form $x \in \mathbb{R}^d \mid x_i = 1$ or $x \in \mathbb{R}^d \mid x_i = 0$. Since $x \in \mathbb{R}^d \mid x_i \leq 1$ and $x \in \mathbb{R}^d \mid x_i \geq 0$ are always valid inequalities in a 0/1-polytope, the new polytope is a face of the old one. It is a basic fact from polytope theory that vertices of faces are vertices of the polytope. Thus, all vertices of the intersection must be 0/1-vectors.

Now, let us show that in a 0/1-polytope, the vertices correspond to the integer points. Consider a 0/1-polytope $P \subseteq \mathbb{R}^d$. Since $P \subseteq [0, 1]^d$, for all $z \in P$ we have $0 \leq z_i \leq 1$ and thus we may suppose to dispose of d such inequalities among those which define P . Now consider $y = (y_1, \dots, y_d)$ an integer solution of P . By definition, $y_i = 1$ or $y_i = 0$ and hence we know that there exists d linearly independent inequalities that are verified as equalities, i.e., y is a vertex. \square

We will solve $\text{ENUM_}\alpha\text{-ORIENTATIONS}$ as follows. First, we will show that $\text{ENUM_}\alpha\text{-ORIENTATIONS}$ reduces to $\text{ENUM_VERTICES-0/1-POLYTOPE}$, the latter being defined as:

ENUM_VERTICES-0/1-POLYTOPE
Input : P a 0/1-polytope
Output : the set of all vertices of P

This reduction boils down to a one-to-one correspondence between the elements of $\mathcal{O}_\alpha(G)$ and the integer solutions of a 0/1-polytope. Thus, by Claim 66, it is enough to prove that $\text{ENUM_INTEGER-POINTS-BOX-POLYTOPE} \in \text{DelayP}$, with:

ENUM_INTEGER-POINTS-BOX-POLYTOPE
Input : P a box-integer-polytope
Output : the set of all integer points of P

Theorem 67. The problem $\text{ENUM_}\alpha\text{-ORIENTATIONS}$ reduces to $\text{ENUM_VERTICES-0/1-POLYTOPE}$.

Proof. We will show that given a graph $G = (V, E)$ and $\alpha : V \rightarrow \mathbb{N}$, there exists a 0/1-polytope P such that the α -orientations of G and $P \cap \{0, 1\}^d$ correspond. In other words, there is a bijection between the elements of $\mathcal{O}_\alpha(G)$ and the integral solutions of 0/1-polytope P . By Claim 66, we will then be able to conclude.

Let G be some graph $G = (V, E)$, α some fixed vector and D_0 some arbitrary orientation of G . Denote by A_{D_0} the set of arcs of D_0 and $\delta_{D_0}^+ = \alpha_0$. Let us show that there is a one-to-one correspondence between $\mathcal{O}_\alpha(G)$ and the integral solutions of the polytope (4.9) below.

$$(4.9) \left\{ \begin{array}{l} \forall v \in V, \sum_{\substack{a=(u,v) \\ a \in A_{D_0}}} y(a) - \sum_{\substack{b=(v,w) \\ b \in A_{D_0}}} y(b) \leq \alpha(v) - \alpha_0(v) \quad (4.9a) \\ \forall v \in V, - \sum_{\substack{a=(u,v) \\ a \in A_{D_0}}} y(a) + \sum_{\substack{b=(v,w) \\ b \in A_{D_0}}} y(b) \leq \alpha(v) - \alpha_0(v) \quad (4.9b) \\ \forall a \in E, y(a) \leq 1 \quad (4.9c) \\ \forall a \in E, y(a) \geq 0, \quad (4.9d) \end{array} \right.$$

Let $D \in \mathcal{O}_\alpha(G)$ and let us show that we can associate with D an integral solution of (4.9). We build a function

$$y : A_{D_0} \rightarrow \{0, 1\}$$

$$y(a) = \begin{cases} 0 & \text{if } a \text{ has the same direction in } D_0 \text{ and } D \\ 1 & \text{otherwise} \end{cases}$$

Observe that y verifies $\sum_{\substack{a=(u,v) \\ a \in A_{D_0}}} y(a) - \sum_{\substack{b=(v,w) \\ b \in A_{D_0}}} y(b) = \alpha(v) - \alpha_0(v)$. In fact we have:

$$\begin{aligned} \alpha(v) - \alpha_0(v) &= \left(\sum_{\substack{a=(v,w) \\ a \in A_D \\ a \in A_{D_0}}} 1 + \sum_{\substack{a=(v,w) \\ a \in A_D \\ a \notin A_{D_0}}} 1 \right) - \left(\sum_{\substack{b=(v,w) \\ b \in A_{D_0} \\ b \in A_D}} 1 + \sum_{\substack{b=(v,w) \\ b \in A_{D_0} \\ b \notin A_D}} 1 \right) \\ &= \sum_{\substack{a=(v,w) \\ a \in A_D \\ a \notin A_{D_0}}} 1 - \sum_{\substack{b=(v,w) \\ b \in A_{D_0} \\ b \notin A_D}} 1 \\ &= \sum_{\substack{a=(w,v) \\ a \notin A_D \\ a \in A_{D_0}}} 1 - \sum_{\substack{b=(v,w) \\ b \in A_{D_0} \\ b \notin A_D}} 1 \\ &= \sum_{\substack{a=(w,v) \\ a \notin A_D \\ a \in A_{D_0}}} y(a) - \sum_{\substack{b=(v,w) \\ b \in A_{D_0} \\ b \notin A_D}} y(b) \end{aligned}$$

Hence, $y(a)$ for all $a \in A_{D_0}$ is an integral solution of (4.9).

Now, take y an integral solution of (4.9) and let us show that we can associate with y an orientation D that belongs to $\mathcal{O}_\alpha(G)$. We construct D from D_0 as follows: if $y(a) = 0$, then the direction of a is

preserved in D , else if $y(a) = 1$, then the direction of a is reversed in D . Observe that for all $v \in A_D$, we have $\delta_D^+(v) = \alpha(v)$. In fact we have:

$$\begin{aligned} \delta_D^+(v) &= \sum_{\substack{a=(v,u) \\ a \in A_{D_0} \\ a \in A_D}} 1 + \sum_{\substack{a=(v,u) \\ a \notin A_{D_0} \\ a \in A_D}} 1 \\ &= \sum_{\substack{a=(v,u) \\ a \in A_{D_0} \\ a \in A_D}} 1 + \sum_{\substack{a=(u,v) \\ a \in A_{D_0} \\ a \notin A_D}} 1 \\ &= \sum_{\substack{a=(v,u) \\ a \in A_{D_0} \\ a \in A_D}} 1 + \sum_{\substack{a=(u,v) \\ a \in A_{D_0}}} y(a) \end{aligned}$$

Since y is an integral solution of (4.9), we know that

$$\sum_{\substack{a=(u,v) \\ a \in A_{D_0}}} y(a) = \alpha(v) - \alpha_0(v) + \sum_{\substack{b=(v,w) \\ b \in A_{D_0}}} y(b)$$

Thus we have:

$$\delta_D^+(v) = \sum_{\substack{a=(v,u) \\ a \in A_{D_0} \\ a \in A_D}} 1 + \alpha(v) - \alpha_0(v) + \sum_{\substack{b=(v,w) \\ b \in A_{D_0}}} y(b)$$

Finally, because $\sum_{\substack{a=(v,u) \\ a \in A_{D_0} \\ a \in A_D}} 1 + \sum_{\substack{b=(v,w) \\ b \in A_{D_0}}} y(b) = \alpha_0(v)$, we can conclude that $\delta_D^+(v) = \alpha(v)$.

Hence, $D \in \mathcal{O}_\alpha(G)$, this completes the proof of the bijection.

With some algebra calculations, we can prove that polytope (4.9) is a 0/1-polytope. By definition the polytope (4.9) is contained in $[0, 1]^d$. Now the defining matrix A is the incidence matrix of a directed graph and therefore totally unimodular, see [90]. Moreover, the right-hand side of A are integral value. The calculation of the vertices boils down to the resolution of a system of Cramer. With a totally unimodular matrix and an integer second member, Cramer's rules gives us integer vertices. We can conclude that the polytope (4.9) is an integer polytope contained in $[0, 1]^d$, i.e., a 0/1-polytope.

We have seen that there is a bijection between the elements of $\mathcal{O}_\alpha(G)$ and the integral solutions of the 0/1-polytope (4.9). By Claim 66, we know that the integral solutions of (4.9) are exactly the vertices. This completes the proof of Theorem 67. □

We saw that `ENUM_α-ORIENTATIONS` reduces to `ENUM_VERTICES-0/1-POLYTOPE`. More precisely we saw that there is a bijection between the integer solutions of a 0/1-polytope and the α -orientations. Since a 0/1-polytope is a box-integer polytope, we are left we the enumeration of the integer points of a box-integer polytope. We will show the following result.

Theorem 68. `ENUM_INTEGER-POINTS-BOX-POLYTOPE` belongs to complexity class DelayP.

We will give two slightly different algorithms for solving `ENUM_INTEGER-POINTS-BOX-POLYTOPE`. In the first algorithm, no box is needed as input, the algorithm determines it by itself and its size will be reflected in the running time analysis. The second algorithm requires as input the bounding box of the polytope and yields the Theorem 68 above.

ENUM_INTEGER-POINTS-BOX-POLYTOPE without the box as input We can use the flashlight backtrack search method. The algorithm generating all possible integer points takes as input a box-integer polytope $P \subseteq \mathbb{R}^d$, a first arbitrary integer point $x = (x_1, \dots, x_d)$ and a cursor $0 \leq p \leq d$, which is initially set to 0. The algorithm modifies step by step x into a new integer point of P . At each recursive call, a coordinate $x_{p+1} \in x$ that has not been considered yet (i.e., each coordinate x_i with $i \leq p$ is already fixed) will be fixed with the three possible cases to consider:

- the integer points y with $y_{p+1} = x_{p+1}$: this branch is known to be non-empty since x is a possible extension, thus we recurse with x and $p + 1$
- the integer points y with $y_{p+1} < x_{p+1}$: we check if $P \cap \{y \mid y_{p+1} = x_{p+1} - 1\} \neq \emptyset$. Since P is a box-integer polyhedron the intersection of P and the box $\{y \mid y_{p+1} = x_{p+1} - 1\}$ is an integer polyhedron. This condition ensures that there exist integer points in $P \cap \{y \mid y_{p+1} = x_{p+1} - 1\}$ and thus that this branch will give rise to a solution. Thus, if $P \cap \{y \mid y_{p+1} = x_{p+1} - 1\} \neq \emptyset$, there are two cases to consider:
 - the integer points y with $y_{p+1} = x_{p+1} - 1$: we recurse with the new integer point $(x_1, \dots, x_{p+1} - 1, \dots, x_d)$ and cursor $p + 1$
 - the integer points y with $y_{p+1} < x_{p+1} - 1$: we pursue the decreasing of x_{p+1} as long as possible
- the integer points y with $y_{p+1} > x_{p+1}$: we check if $P \cap \{y \mid y_{p+1} = x_{p+1} + 1\} \neq \emptyset$. Analogously to the item before, if $P \cap \{y \mid y_{p+1} = x_{p+1} + 1\} \neq \emptyset$, we consider two cases:
 - the integer points y with $y_{p+1} = x_{p+1} + 1$: we recurse with the new integer point $(x_1, \dots, x_{p+1} + 1, \dots, x_d)$ and cursor $p + 1$
 - the integer points y with $y_{p+1} > x_{p+1} + 1$: we pursue the increasing of x_{p+1} as long as possible.

We will use the notation e_p for the vector whose all components are 0 except the p th which is 1.

Algorithm 15: Enumeration of the integer points of a box-integer polytope without the box as input

Input: a box-integer polytope $P \subseteq \mathbb{R}^d$

Output: all integer points of P

```

1 begin
2   if  $P \neq \emptyset$  then
3     Take  $z \in P \cap \mathbb{Z}^d$  an integer point
4     Generate ( $P, z, 0$ )
5 end

6 Function Generate (a box-integer polytope  $P \subseteq \mathbb{R}^d, x = (x_1, \dots, x_d) \in P \cap \mathbb{Z}^d, p \in \{0, \dots, d-1\}$ ):
7   if  $p < d$  then
8     Increase ( $P, x, p$ )
9     Decrease ( $P, x, p$ )
10    Generate ( $P \cap \{y \mid y_{p+1} = x_{p+1}\}, x, p+1$ )
11  else
12    Output  $x$ 
13 end

14 Function Decrease (a box-integer polytope  $P \subseteq \mathbb{R}^d, x = (x_1, \dots, x_d) \in P \cap \mathbb{Z}^d, p \in \{0, \dots, d-1\}$ ):
15   if  $P \cap \{y \mid y_{p+1} = x_{p+1} - 1\} \neq \emptyset$  then
16     Decrease ( $P, x - e_{p+1}, p$ )
17     Generate ( $P \cap \{y \mid y_{p+1} = x_{p+1} - 1\}, x - e_{p+1}, p+1$ )
18   end
19 end

20 Function Increase (a box-integer polytope  $P \subseteq \mathbb{R}^d, x = (x_1, \dots, x_d) \in P \cap \mathbb{Z}^d, p \in \{0, \dots, d-1\}$ ):
21   if  $P \cap \{y \mid y_{p+1} = x_{p+1} + 1\} \neq \emptyset$  then
22     Increase ( $P, x + e_{p+1}, p$ )
23     Generate ( $P \cap \{y \mid y_{p+1} = x_{p+1} + 1\}, x + e_{p+1}, p+1$ )
24   end
25 end

```

Theorem 69. Let $P \subseteq \mathbb{R}^d$ be a box-integer polytope and let $x = (x_1, \dots, x_d)$ be an integer point of P . For all p such that $1 \leq p \leq d$, function **Generate** (P, x, p) generates all integer points that coincide with x on coordinates x_1 to x_p . For $p = 0$ function **Generate** (P, x, p) generates all integer points. Furthermore, Algorithm 15 does not repeat twice the same solution.

Proof. We prove now the completeness of our algorithm and show that there is no redundancy.

Completeness We prove the theorem by descending induction on p . If $p = d$, then **Generate**(P, x, p) = **Generate**(P, x, d) = $\{x\}$, Theorem 69 is verified. Let us show that it is also the case for all $p < d$. Suppose that for some $1 \leq p \leq d$ we have the following: for all box-integer polytope $P \subseteq \mathbb{R}^d$ and for all integer point $x \in P$, **Generate**(P, x, p) = $\{y \mid y \text{ is an integer point of } P \text{ and } y_i = x_i \text{ for all } i \leq p\}$. Let us show that it is the case for $p-1$ too (i.e., if $p > 1$, then **Generate**($P, x, p-1$) = $\{y \mid y \text{ is an integer point of } P \text{ and } y_i = x_i \text{ for all } i \leq p-1\}$ and if $p = 1$, then **Generate**($P, x, p-1$) = $\{y \mid y \text{ is an integer point of } P\}$). Consider $y = (y_1, \dots, y_d)$ an integer point of P such that $y_i = x_i$ for all $i \leq p-1$. We will show that y is generated

by function $\text{Generate}(P, x, p - 1)$. Take y_p , there are two cases to consider: $y_p = x_p$ or $y_p \neq x_p$. In the first case, we know by induction hypothesis that y is generated by function $\text{Generate}(P, x, p)$. Because in the tree built by our algorithm, branch $\text{Generate}(P, x, p)$ is a child of $\text{Generate}(P, x, p - 1)$, we conclude that y is generated by $\text{Generate}(P, x, p - 1)$. Now consider the case where $y_p \neq x_p$. Suppose that $y_p < x_p$, let us say that $y_p = x_p - \alpha$. Because $y \in P$, we have $P \cap \{z \mid z_p = x_p - \alpha\} \neq \emptyset$. Hence, by Observation 2 we know that for $j \leq \alpha$, $P \cap \{z \mid z_p = x_p - j\} \neq \emptyset$. Consider the case where $j = 1$. Function Decrease gives rise to two branches: $\text{Generate}(P, x - e_p, p)$ and $\text{Decrease}(P, x - e_p, p - 1)$. Either $y_p = x_p - 1$ and then y coincides with $x - e_p$ on $\{1, \dots, p\}$. By induction hypothesis y is generated by $\text{Generate}(P, x - e_p, p)$, which is a child of $\text{Generate}(P, x, p - 1)$. Or $y_p < x_p - 1$ and then we repeat the decreasing process with $\text{Decrease}(P, x - e_p, p - 1)$ until we obtain $y_p = x_p$. Finally, notice that the case $y_p > x_p$ can be proven the same way with the help of function Increase instead of Decrease .

Irredundancy Notice that each node built by this algorithm gives rise to branches that are disjoint. Function Generate gives place to nodes with three children, one where we decrease the value of coordinate x_i , one where we fix it and one where we increase it. Function Decrease (resp. Increase) generates nodes with only two sons, one where we keep trying to decrease (resp. increase) the value of coordinate x_i and one where we fix it.

Figures 4.10 and 4.11 illustrate the recursion tree built by Algorithm 15. In Figure 4.10 each node is labeled with the current test of feasibility. Figure 4.11 illustrates the way the integer point taken as input is modified into a new integer point. The coordinates in red are the one that are fixed. \square

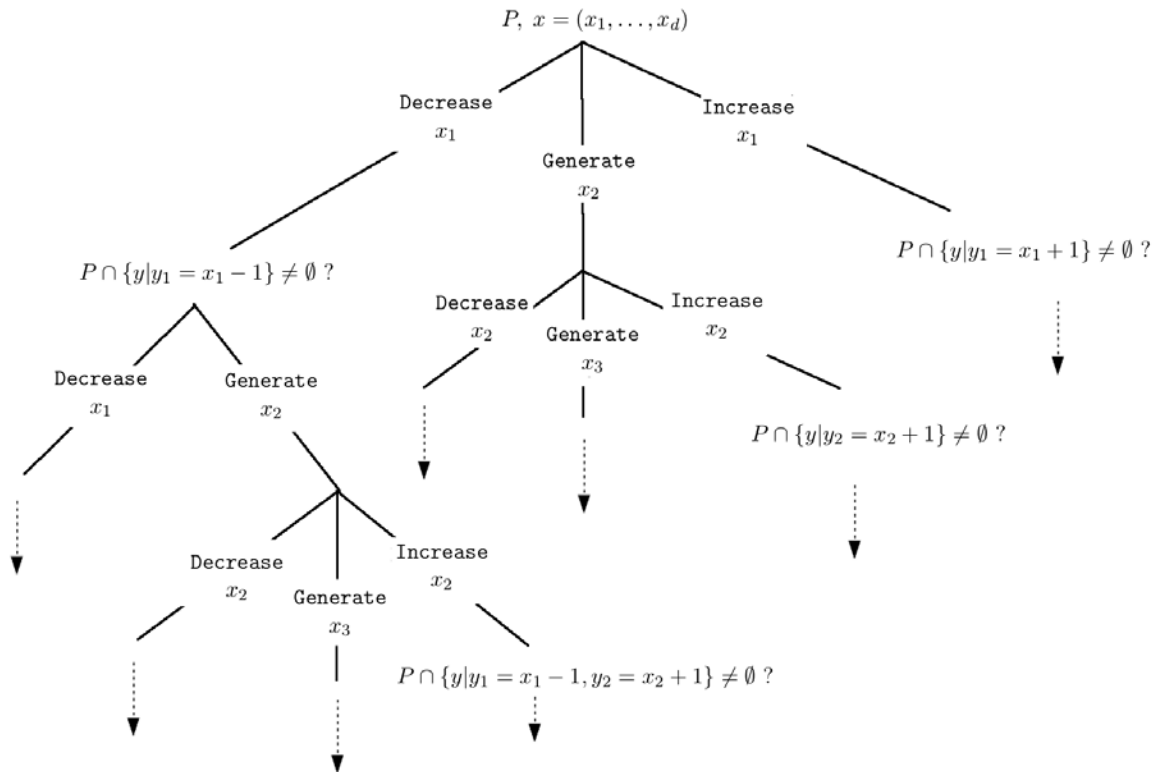


Figure 4.10: Recursion Tree (1) for Algorithm 15

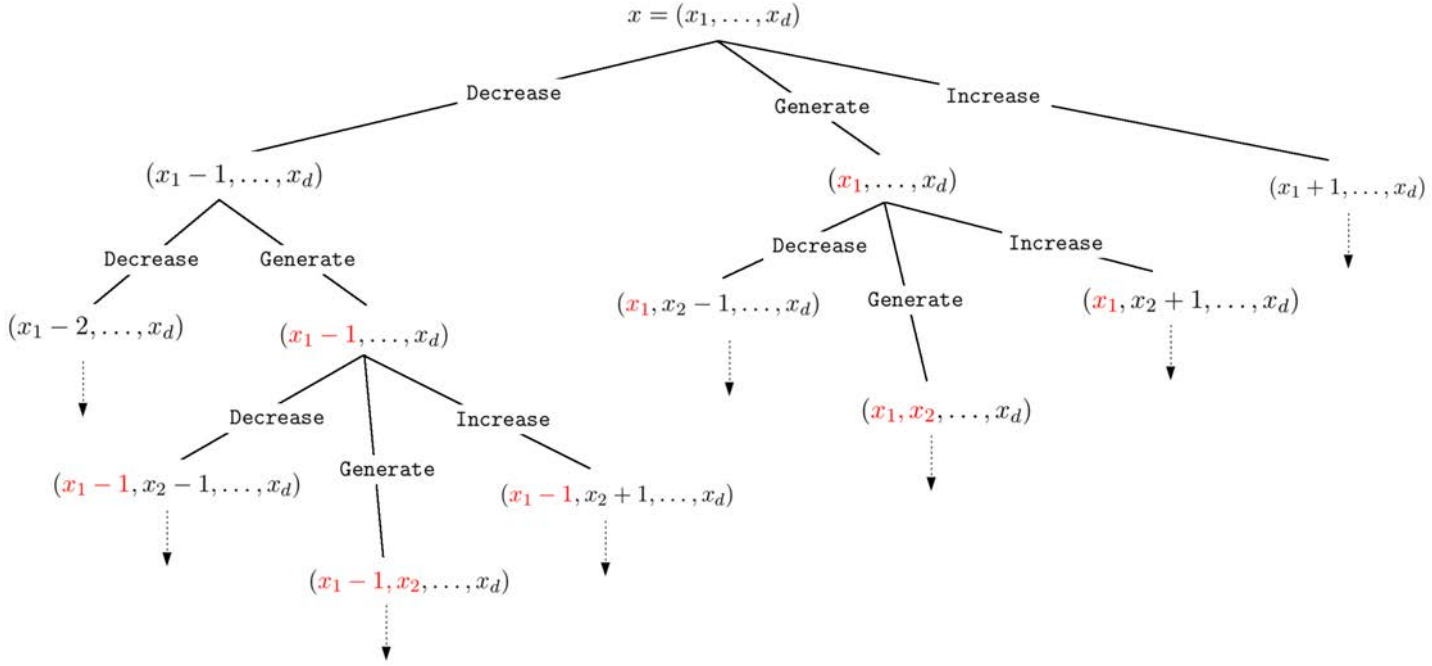


Figure 4.11: Recursion Tree (2) for Algorithm 15

Now let us take a look at the complexity. In each call of **Increase** or **Decrease** it has to be checked if the polytope is feasible. About the feasibility of a polytope, there is a result by Khachiyan:

Theorem 70. [90, Theorem 13.4] *Systems of rational linear inequalities, and linear programming problems with rational data, can be solved in polynomial time. More precisely, if A is a $q \times d$ matrix, $b \in \mathbb{Z}^q$ and T the maximum absolute value of the entries A and b , then a linear programming problem can be solved in total time $O(d^8 \cdot \log^2 T)$.*

Khachiyan's algorithm for obtaining the above result uses the ellipsoid method; it is the first worst-case polynomial-time algorithm ever found for linear programming. Although the ellipsoid method is theoretically efficient, computational experiments with the method are very discouraging. Some improvements have been proposed since [81, 92]. Here, we will not focus on the best complexity.

At each node, the complexity of Algorithm 15 is in $O(d^8 \cdot \log^2 T)$. The depth of the recursion tree depends on the bounding box $\mathbf{B}(l, u)$ of polytope P which is in $O(d(l, u)) = O(\sum_{p=0}^d (u_{p+1} - l_{p+1}))$.

Observe that Algorithm 15 has to use a separate method for finding a first integer point $z \in P$. This can be done with Algorithm 16 below, that works in the following way: for $P \subseteq \mathbb{R}^d$ and $p = 1$ to d , at each step the ellipsoid method is used to find a feasible solution that maximizes coordinate x_p . When a feasible solution is found with x_p maximal, we know that x_p is integer, otherwise there would exist some non integer vertices.

Algorithm 16: Finding an integer point of a box-integer polytope

Input: a box-integer polytope $P \subseteq \mathbb{R}^d$
Output: an integer point of P

```

1 begin
2   if  $P \neq \emptyset$  then
3     Generate ( $P, 1$ )
4 end

5 Function Generate (a box-integer polytope  $P \subseteq \mathbb{R}^d$ ,  $1 \leq p < d$  an integer):
6   if  $p \leq d$  then
7     Find  $x$  a feasible solution of  $P$  which maximizes  $x_p$ 
8     Generate ( $P \cap \{y \in \mathbb{R}^d \mid y_p = x_p\}, p + 1$ )
9   else
10    Output  $P$ 
11 end

```

Thus, this pre-processing step takes a time in $d \times (d^8 \cdot \log^2 T)$.

We can conclude that the total delay of Algorithm 15 is in

$$O((d^9 \cdot \log^2 T) + \mathbf{B}(l, u)(d^8 \cdot \log^2 T)) = O((d^9 \cdot \log^2 T) + \sum_{i=1}^d (u_i - l_i)(d^8 \cdot \log^2 T))$$

Remark 5. A bounding box can be easily calculated by maximizing and minimizing once on each coordinate.

ENUM_INTEGER-POINTS-BOX-POLYTOPE with the box as input A variant of Algorithm 15 consists in taking as input the bounding box $\mathbf{B}(l, u) \in \mathbb{Z}^d$ associated to the polytope P . Hence, instead of trying to increase and decrease each coordinate of some integer point taken as input, we check the feasibility only with the coordinates in between l and u , i.e., for all $i \in \{1, \dots, d\}$ we check for each y_i such that $l_i \leq y_i \leq u_i$ if $P \cap \{y_i\}$ is feasible or not. This approach is an instance of constructing an enumeration algorithm out of EXTSOL.

We will use the notation $\mathbf{0}$ for the vector whose all components are 0.

Algorithm 17: Enumeration of the integer points of a box-integer polytope with the box as input

Input: a box-integer polytope $P \subseteq \mathbb{R}^d$, the box $\mathbf{B}(l, u)$ of P

Output: all integer points of P

```

1 begin
2   if  $P \neq \emptyset$  then
3     Generate ( $P, \mathbf{B}(l, u), \mathbf{0}, 0$ )
4 end

5 Function Generate (a box-integer polytope  $P \subseteq \mathbb{R}^d$ , the box  $\mathbf{B}(l, u)$  of  $P$ ,  $x \in \mathbb{R}^d$ ,  $0 \leq p < d$  an integer):
6   if  $p < d$  then
7     for  $j = l_{p+1}$  to  $u_{p+1}$  do
8       if  $P \cap \{y_{p+1} = j\} \neq \emptyset$  then
9         Generate ( $P \cap \{y_{p+1} = j\}, \mathbf{B}(l, u), x + j \cdot e_{p+1}, p + 1$ )
10      else
11        Output  $x$ 
12 end

```

The proof of this algorithm is similar to the one of Algorithm 15. Figure 4.12 below illustrates its recursion tree.

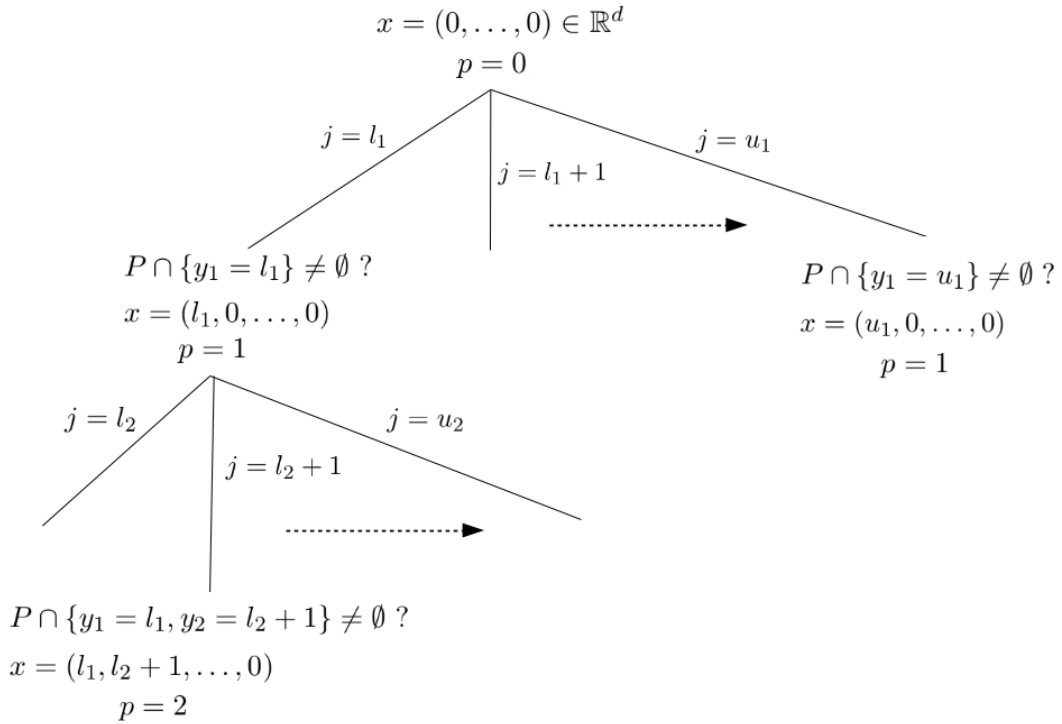


Figure 4.12: Recursion Tree for Algorithm 17

The complexity of Algorithm 17 is similar to the one of Algorithm 15 that doesn't take the box as input. The difference is that no first integer point is needed as pre-processing step and this time, disposing of the box $\mathbf{B}(l, u)$ of P , we are able to bound precisely the depth of the tree. The delay of Algorithm 17 is in $O(d(l, u)(d^8 \cdot \log^2 T))$.

Observe that Algorithm 17 specializes back exactly to the enumeration of vertices of a 0/1-polytope. If y_1, \dots, y_d denotes the variables of a 0/1-polytope P and if for example $P' = P \cap \{y_1 = 1\}$, notice that if P' is feasible, then P' contains a 0/1-vertex. In fact, if P' is feasible but does not contain a 0/1-vertex, then, all vertices of P have $y_1 = 0$, and so every solution of P too. This is in contradiction with the fact that P' is feasible. Hence, for the enumeration of the vertices of a 0/1-polytope, Algorithm 17 becomes:

Algorithm 18: Algorithm 17 for ENUM_VERTEX-0/1-POLYTOPE	
Input:	a 0/1-polytope $P \in [0, 1]^d$
Output:	all vertices of P
1	begin
2	if $P \neq \emptyset$ then
3	Generate ($P, \mathbf{0}, 0$)
4	end
5	Function Generate (a 0/1-polytope $P \in [0, 1]^d$, $x \in \mathbb{R}^d$, an integer $0 \leq p < d$):
6	if $p < d$ then
7	if $P \cap \{y_{p+1} = 1\} \neq \emptyset$ then
8	Generate ($P \cap \{y_{p+1} = 1\}, x + e_{p+1}, p + 1$)
9	if $P \cap \{y_{p+1} = 0\} \neq \emptyset$ then
10	Generate ($P \cap \{y_{p+1} = 0\}, x, p + 1$)
11	else
12	Output x
13	end

4.4.2 Flashlight backtrack search via cycle flips

Another way for enumerating $\mathcal{O}_\alpha(G)$ is a flashlight backtrack search using Lemma 55. This method is simpler to state and more efficient. The algorithm takes as an input a directed graph $D = (V, A) \in \mathcal{O}_\alpha(G)$ and a set of *fixed* arcs $F \subseteq A$. Initially D is an arbitrary α -orientation and $F = \emptyset$. The algorithm recursively constructs all orientations of $\mathcal{O}_\alpha(G)$ such that at each recursive call all possible α -orientations contain the current F . At each recursive call, an arc $a \notin F$ is fixed with the two possible cases to consider:

- The α -orientations with $a = (u, v)$: this branch is known to be non-empty since D is a possible extension, thus we recurse with D and $F \cup a$;
- The α -orientations with $a^- = (v, u)$: by Lemma 55, there exists an orientation in $\mathcal{O}_\alpha(G)$ with a^- if and only if a^- belongs to some directed cycle of D . Thus, if there is a directed path P in $D \setminus F$ from v to u , then the directed cycle a, P is reversed and we recurse with the new α -orientation $D^{a \cup P}$ and $F \cup a^-$.

When all the arcs are fixed, we output the current orientation.

Algorithm 19: Flashlight backtrack search for α -orientations

Input: a graph $G = (V, E)$ and $\alpha : V \rightarrow \mathbb{N}$

Output: all elements of $\mathcal{O}_\alpha(G)$

```

1 begin
2   if there exists  $D = (V, A) \in \mathcal{O}_\alpha(G)$  then
3     Fix an arbitrary linear order on  $A$ ;
4     EnOPODS( $D, \emptyset$ );
5   end
6   Function EnOPODS( $D, F$ ):
7     if  $F \neq A$  then
8       Take the smallest  $a = (u, v) \in A \setminus F$ ;
9       EnOPODS( $D, F \cup \{a\}$ );
10      if  $D \setminus F$  has a directed path  $P$  from  $v$  to  $u$  then
11        EnOPODS( $D^{a \cup P}, F \cup \{a^-\}$ );
12      else
13        Output  $D$ ;
14    end

```

Algorithm 19 takes an undirected graph $G = (V, E)$ as input and ends after the traversal of E . After finding one initial α -orientation D , it just consists of recursive calls of the function $\text{EnOPODS}(D, F)$. See the example developed in Figure 4.13 below.

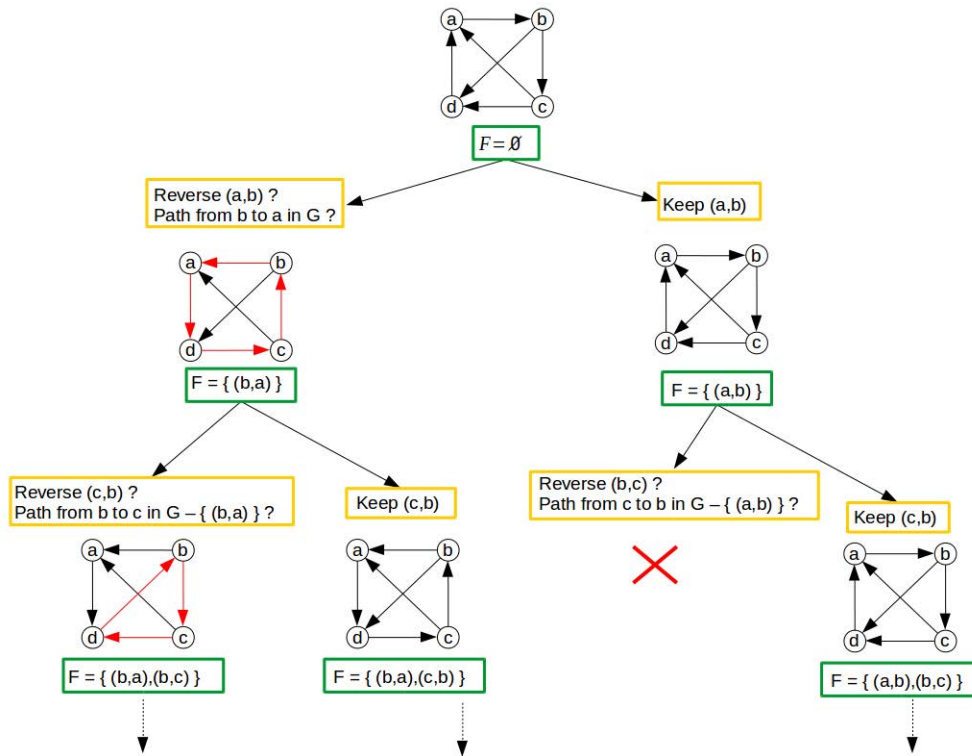


Figure 4.13: Example of application of Algorithm 19

Theorem 71. *Let $D \in \mathcal{O}_\alpha(G)$ and $F \subseteq A$. The function $\text{EnOPDS}(D, F)$ enumerates each α -orientation that coincides with D on F . Furthermore, Algorithm 19 enumerates $\mathcal{O}_\alpha(G)$ without repetition and with a delay of $O(m^2)$.*

Proof. We prove the completeness, i.e., for a fixed α all α -orientations are enumerated; then we show that there is no redundancy and finally we analyse its complexity.

Completeness We start by proving by induction on $|A \setminus F|$ that each of the claimed α -orientations is enumerated. If $|A \setminus F| = 0$, then $\text{EnOPDS}(D, F) = \text{EnOPDS}(D, A) = \{D\}$ and we are done. Let now $|A \setminus F| > 0$ and $a = (u, v) \in A \setminus F$. By induction hypothesis $\text{EnOPDS}(D, F \cup \{a\})$ enumerates each α -orientation that coincides with D on $F \cup \{a\}$ and $\text{EnOPDS}(D^{a \cup P}, F \cup \{a^-\})$ enumerates each α -orientation that coincides with $D^{a \cup P}$ on $F \cup \{a^-\}$. Since $a \cup P$ is a directed cycle, the digraph $D^{a \cup P}$ also is an α -orientation of G by Lemma 55 that fixes F , since $P \cap F = \emptyset$.

Let us proof that if there is no directed path P from u to v in $D \setminus F$, then there exists no α -orientation fixing F and reversing a . By contraposition, suppose that D' is an α -orientation that coincides with D on F but differs on a . Then by Lemma 55, there is a set of arc disjoint directed cycles in D' whose union is $D' \setminus D$. Since both digraphs coincide on F , these cycles are disjoint from F . Since both digraphs differ on a , one of the directed cycles C contains a^- in D' . Thus, the path $P = (C \setminus \{a^-\})^-$ is a directed path in $D \setminus F$ from v to u .

Irredundancy Clearly, both sets generated by $\text{EnOPDS}(D, F \cup \{a\})$ and $\text{EnOPDS}(D^{a \cup P}, F \cup \{a^-\})$ are disjoint since they differ on the orientation of a . Each node built by this algorithm gives rise to branches that are disjoint. In one branch, if possible, the considered arc is reversed and in the other one the direction is not changed. Already fixed arcs will not be considered a second time in the rest of the algorithm. Hence, our algorithm does not generate twice the same solution.

Complexity Observe that Algorithm 19 has to use a separate method for finding a first element $D \in \mathcal{O}_\alpha(G)$. It is well-known that this problem can be reduced to a flow-problem see e.g. [34]. Therefore, this pre-processing step can be done in $O(mn)$ time, see [84].

The depth of the recursion tree is bounded by m . In each recursion step the algorithm only needs to check the presence of a directed path, which can be done by a single BFS from the source vertex u towards the target v . In general the complexity of a BFS algorithm is $O(m + n)$, however in our case the BFS tree will be constructed only on the strongly connected component of D containing u . Since in a connected graph $m \geq n - 1$, a breadth-first search initiated on a connected component has complexity $O(m + n) = O(m)$. Hence, the total delay of Algorithm 19 is bounded by $O(m^2)$. □

4.5 k -connected outdegree sequences via path flips

In this section we will present an algorithm to enumerate the possible outdegree sequences among the k -connected orientations of a graph G .

A direct corollary of Lemma 64 is the following (see also [40]).

Corollary 72. *If $D, D' \in \mathcal{O}_\alpha(G)$, then D is k -connected if and only if D' is k -connected.*

Given $G = (V, E)$, Corollary 72 allows to define a function $\alpha : V \rightarrow \mathbb{N}$ to be k -connected if there is some k -connected $D \in \mathcal{O}_\alpha(G)$. In this case we will sometimes call α a k -connected outdegree sequence. Having

in mind that we want to enumerate all k -connected orientations of G and already are able to enumerate $\mathcal{O}_\alpha(G)$ for any given α , we are left with enumerating the k -connected outdegree sequences.

ENUM_ k -CONNECTED-OUTDEGREE-SEQUENCES	
<i>Input</i> :	a graph $G = (V, E)$, $k \in \mathbb{N}$
<i>Output</i> :	the k -connected functions $\alpha : V \rightarrow \mathbb{N}$

Theorem 73. *The problem ENUM_ k -CONNECTED-OUTDEGREE-SEQUENCES can be solved with a delay in $O(knm^2)$. In particular, the problem belongs to DelayP.*

To show this theorem, we will present two different methods to generate the possible outdegree sequences among the k -arc-connected orientations of a graph G . The first one, stated in Subsection 4.5.1, is an illustration of the reverse search technique; the second one, stated in Subsection 4.5.2, is more efficient and is a flashlight backtrack search similar to Algorithm 15 for enumerating the integer points of a box-integer polytope. The latter being the more efficient and yielding the desired complexity $O(knm^2)$ of Theorem 73 above, it will be the one retained for the final algorithm enumerating all k -connected orientations.

Both of these techniques rely on Theorem 54 and more precisely on Lemma 58 in order to be able to move from one outdegree sequence to another. Here in order to change the outdegree sequence of D we will reverse a directed path P_{uv} from u to v and thus only increase $\delta_D^+(u)$ by one and decrease $\delta_D^+(v)$ by one (recall Lemma 57 and Claim 56). We have to check how reversing paths affects the number of arc-disjoint directed paths between pairs of vertices. The following will be useful:

Lemma 74. *Let P_{uv} be a directed path from a vertex u to a vertex v in some orientation D . Then for all vertices u', v' , we have $\lambda_{DP_{uv}}(u', v') \geq \min(\lambda_D(u, v) - 1, \lambda_D(u', v'))$. Furthermore, for $u' = u$ and $v' = v$, we have $\lambda_{DP_{uv}}(u, v) = \lambda_D(u, v) - 1$.*

Proof. With Menger's theorem (Theorem 53) and Claim 56, Lemma 74 can be easily proved:

$$\begin{aligned}
 \lambda_{DP_{uv}}(u', v') &= \min\{\delta_{DP_{uv}}^+(X) \mid X \subseteq V, u' \in X, v' \notin X\} && \text{(Theorem 53)} \\
 &\geq \min\{\min\{\delta_D^+(X) - 1 \mid X \subseteq V, u', u \in X, v', v \notin X\}, && \text{(Claim 56)} \\
 &\quad \min\{\delta_D^+(X) \mid X \subseteq V, u', u, v \in X \text{ and } v' \notin X \text{ or } u' \in X \text{ and } v', u, v \notin X\}, \\
 &\quad \min\{\delta_D^+(X) + 1 \mid X \subseteq V, u', v \in X, v', u \notin X\}\} \\
 &\geq \min\{\lambda_D(u, v) - 1, \lambda_D(u', v')\} && \text{(Theorem 53)}
 \end{aligned}$$

Now consider the case where $u' = u$ and $v' = v$. We have:

$$\begin{aligned}
 \lambda_{DP_{uv}}(u, v) &= \min\{\delta_{DP_{uv}}^+(X) \mid X \subseteq V, u' \in X, v' \notin X\} && \text{(Theorem 53)} \\
 &= \min\{\delta_D^+(X) - 1 \mid X \subseteq V, u' \in X, v' \notin X\} && \text{(Claim 56)} \\
 &= \lambda_D(u, v) - 1 && \text{(Theorem 53)}
 \end{aligned}$$

□

Recall that in a k -arc-connected orientation D , a directed path P is said to be flippable if D^P is k -arc-connected. And we denoted by flippable a pair of vertices (u, v) for which there exists a flippable path P_{uv} from u to v . It turns out that if a path from u to v is flippable, then all of them are. Actually, Lemma 74 implies more than that: if D is a k -arc-connected orientation and u, v two vertices, then each path from u to v can be reversed without breaking the k -arc-connectivity if and only if there exists $k + 1$ disjoint paths in D from u to v .

Corollary 75. *Let D be a k -arc-connected orientation of some graph $G = (V, E)$ and $u, v \in V$.*

1. *If some path from u to v is flippable, then all paths from u to v are flippable;*
2. *(u, v) is flippable if and only if $\lambda_D(u, v) > k$.*

Proof. Consider P_{uv} a flippable path from u to v . First we show (2): let us prove that $\lambda_D(u, v) > k$. Since P_{uv} is flippable, we have $\lambda_{D^{P_{uv}}}(u, v) \geq k$. By Lemma 74, this is equivalent to $\lambda_D(u, v) - 1 \geq k$ and thus we can conclude that $\lambda_D(u, v) > k$.

Now suppose that $\lambda_D(u, v) > k$ and let us prove that the pair (u, v) is flippable, i.e., $D^{P_{uv}}$ is k -arc-connected, i.e., for all u', v' we have $\lambda_{D^{P_{uv}}}(u', v') \geq k$. By Lemma 74, we know that for all u', v' , $\lambda_{D^{P_{uv}}}(u', v') \geq \min(\lambda_D(u, v) - 1, \lambda_D(u', v'))$. Orientation D being k -arc-connected, $\lambda_D(u', v') \geq k$ and because by hypothesis $\lambda_D(u, v) > k$, we have $\lambda_D(u, v) - 1 \geq k$. This completes the proof of (2).

Let us now prove (1). Consider another path P'_{uv} from u to v and let us show that like P_{uv} , it is a flippable path. Again, by Lemma 74, we know that for all u', v' , $\lambda_{D^{P'_{uv}}}(u', v') \geq \min(\lambda_D(u, v) - 1, \lambda_D(u', v'))$. Orientation D being k -arc-connected, $\lambda_D(u', v') \geq k$. Furthermore, by (2) we know that $\lambda_D(u, v) > k$ and thus $\lambda_D(u, v) - 1 > k - 1$. Hence $\lambda_{D^{P'_{uv}}}(u', v') \geq k$, which completes the proof of (1). □

Lemma 76. *Let D be k -connected, it can be decided in time $O(km)$ if (u, v) is flippable.*

Proof. By Lemma 74, we have $\lambda_D(u, v) > k$, i.e., there exists at least $k + 1$ disjoint paths from u to v in D , if and only if the procedure of reversing a path from u to v can be applied $k + 1$ times. Hence, a simple algorithm consists in finding a directed path P_{uv} in D , reverse it and iterate in $D^{P_{uv}}$. We have $\lambda_D(u, v) > k$ if and only if this procedure can be applied $k + 1$ times. Each execution is a BFS, which yields the claimed running time.

Algorithm 20: Checking if a pair (u, v) of vertices is flippable

Input: a graph $G = (V, E)$, an orientation D , $u, v \in V$ and an integer k

Output: is the pair of vertices (u, v) flippable?

```

1 while there exists a path  $P_{uv}$  from  $u$  to  $v$  in  $D$  and COUNT <  $k + 1$  do
2    $D \leftarrow D^{P_{uv}}$ 
3   COUNT  $\leftarrow$  COUNT + 1
4 if COUNT =  $k + 1$  then
5   Output: "Yes"
6 else
7   Output: "No"

```

Only the connected component containing u and v needs to be considered. Recall that in a connected graph $m \geq n - 1$, and thus a breadth-first search has complexity $O(m + n) = O(m)$. Hence, Algorithm 20 above runs in $O(km)$ □

Remark 6. *A natural way of extending Algorithm 20 consists in applying the procedure for all n^2 possible couples of vertices, thus obtaining an algorithm for checking if an orientation is k -arc-connected in $O(n^2km)$.*

4.5.1 Reverse search

Let $G = (V, E)$ be an undirected graph such that $V = \{x_1, \dots, x_n\}$ is ordered. We endow $V \times V$ with the lexicographic order: $(x_i, x_j) \leq (x_{i'}, x_{j'})$ if $i < i'$ or $i = i'$ and $j \leq j'$. We show how to enumerate all k -connected outdegree sequences of some graph G using the *Reverse Search* method.

Definition of the metagraph: We want to construct and to go through a metagraph \mathcal{G} which will give us all k -connected outdegree sequences of G . We will construct our metagraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ as follows:

- The vertices \mathcal{V} are the k -connected outdegree sequences;
- there is an edge from a k -connected outdegree sequence α to another k -connected outdegree sequence α' if there exist two k -arc-connected orientations $D \in \mathcal{O}_\alpha(G)$ and $D' \in \mathcal{O}_{\alpha'}(G)$ such that D and D' differ only by the direction of one path.

In order to construct this metagraph, we first have to define the adjacency oracle and the local search function.

Definition of the adjacency oracle: The adjacency oracle allows us to list the neighbours of a given vertex of \mathcal{G} . In order to define it properly, first observe the following corollary of Lemma 64.

Corollary 77. *Let u, v be two vertices of some graph G . If (u, v) is flippable in some orientation $D \in \mathcal{O}_\alpha(G)$, then (u, v) is flippable in every orientation $D' \in \mathcal{O}_\alpha(G)$.*

Proof. Let (u, v) be a flippable pair in orientation $D \in \mathcal{O}_\alpha(G)$. By Corollary 75, we have $\lambda_D(u, v) > k$ and with Lemma 64, we know that $\lambda_{D'}(u, v) = \lambda_D(u, v) > k$. Thus, again with Corollary 75 we can conclude that (u, v) is flippable in D' . \square

Having Corollary 77 in mind, we can define the adjacency oracle as follows. First, set $\overline{\Delta}(\mathcal{G})$ to be an upper bound of the degree of \mathcal{G} , and $D \in \mathcal{O}_\alpha(G)$. Let $1 \leq p \leq \overline{\Delta}(\mathcal{G})$ and let (x_i, x_j) be the p -th element of $V \times V$.

If (x_i, x_j) is flippable in D , then construct $\delta_{D'}^+$:

- (1) $\delta_{D'}^+(x_i) = \delta_D^+(x_i) - 1$;
- (2) $\delta_{D'}^+(x_j) = \delta_D^+(x_j) + 1$;
- (3) for all $l \neq i, j$, $\delta_{D'}^+(x_l) = \delta_D^+(x_l)$;

and

$$Adj(\alpha, p) = \delta_{D'}^+$$

else

$$Adj(\alpha, p) = 0.$$

Let us prove that the adjacency oracle is properly defined.

Proof. (adjacency oracle) We have to prove the two points below.

1. if $Adj(\alpha, p) = Adj(\alpha, p') \neq 0$, then $p = p'$;
2. for all $\alpha \in \mathcal{V}$, $\{Adj(\alpha, p) \mid Adj(\alpha, p) \neq 0, 1 \leq p \leq \overline{\Delta}(\mathcal{G})\}$ is exactly the set of neighbours of α .

Let $Adj(\alpha, p) = \alpha'$. Let (x_i, x_j) be the p -th element of $V \times V$. By the definition of Adj , we know that α' differs from α only on the two indices i and j . Now let p' be the index of the couple $(x_{i'}, x_{j'})$. If $p \neq p'$, then $(x_i, x_j) \neq (x_{i'}, x_{j'})$, and so by construction of our adjacency oracle, $Adj(\alpha, p) \neq Adj(\alpha, p')$.

Now let α and α' be two adjacent vertices of our metagraph \mathcal{G} . By the definition of \mathcal{E} , there exist $D \in \mathcal{O}_\alpha(G)$ and $D' \in \mathcal{O}_{\alpha'}(G)$ such that D' can be obtained from D by reversing one path from a vertex x_i to a vertex x_j . By Lemma 57, D' satisfies conditions (1) to (3) of our adjacency oracle. If p is the index of (x_i, x_j) , then $\alpha' \in Adj(\alpha, p)$. □

Definition of the local search: Recall that the local search is a procedure that enables us to reach from any vertex of the metagraph a predefined subset of vertices of \mathcal{G} called the sink. This procedure works with a function f which allows a vertex to identify its successor.

Recall that the distance between the vectors α and α' is defined as $d(\alpha, \alpha') = \sum_{i=1}^n |\alpha_i - \alpha'_i|$.

We define our local search (\mathcal{G}, S, f) such that $S \subseteq \mathcal{V}$ and $f : \mathcal{V} \setminus S \rightarrow \mathcal{V}$ by

- $S = \alpha_0$ with $\alpha_0 = \delta_{D_0}^+$ and D_0 a first k -arc-connected orientation of G ;
- $f(\alpha) = \alpha'$ such that
 - $\alpha' \in Adj(\alpha)$;
 - $d(\alpha', \alpha_0) = \min\{d(\alpha'', \alpha_0) \mid \alpha'' \in Adj(\alpha)\}$.

In the case of several minima we choose $\alpha' \in Adj(\alpha, p)$ with the smallest p .

Proof. We have to prove that there exists $k \in \mathbb{N}$ such that $f^k(\alpha) = \alpha_0$. Let the sequence $(u_l) = (d(f^l(\alpha), \alpha_0))$. For all l we have $u_l \in \mathbb{N}$ and it is enough to prove that (u_l) is strictly decreasing.

Let us first compare $d(f(\alpha), \alpha_0)$ and $d(\alpha, \alpha_0)$. Take $D \in \mathcal{O}_\alpha(G)$ and $D_0 \in \mathcal{O}_{\alpha_0}(G)$. Because D and D_0 are two k -arc-connected orientations with different outdegree sequences, by Lemma 58

- there exists $x_i \in V$ such that $\delta_D^+(x_i) > \delta_{D_0}^+(x_i)$;
- there exists $x_j \in V$ such that $\delta_D^+(x_j) < \delta_{D_0}^+(x_j)$;

and reversing a path from x_i to x_j gives us a new orientation D' still k -arc-connected and such that

- $\delta_{D'}^+(x_i) = \delta_D^+(x_i) - 1$;
- $\delta_{D'}^+(x_j) = \delta_D^+(x_j) + 1$;
- for all $l \neq i, j$ we have $\delta_{D'}^+(x_l) = \delta_D^+(x_l)$.

So we have

- $|\delta_{D'}^+(x_i) - \delta_{D_0}^+(x_i)| < |\delta_D^+(x_i) - \delta_{D_0}^+(x_i)|$;

- $|\delta_{D'}^+(x_j) - \delta_{D_0}^+(x_j)| < |\delta_D^+(x_j) - \delta_{D_0}^+(x_j)|$;
- for all $l \neq i, j$ we have $|\delta_{D'}^+(x_l) - \delta_{D_0}^+(x_l)| = |\delta_D^+(x_l) - \delta_{D_0}^+(x_l)|$.

As a consequence we have

$$d(\delta_{D'}^+, \alpha_0) < d(\alpha, \alpha_0)$$

with $\delta_{D'}^+ \in \text{Adj}(\alpha)$. So by the definition of $f(\alpha)$ we have

$$d(f(\alpha), \alpha_0) < d(\alpha, \alpha_0).$$

We have shown that $u_1 < u_0$, and it can be shown in the same way that if $f^l(\alpha) \neq \alpha_0$, then $u_{l+1} < u_l$. We conclude that there exists k such that $u_k = 0$. □

Analysis of the complexity Let t_{Adj} (resp. t_f) be the time to calculate $\text{Adj}(\alpha, p)$ (resp. $f(\alpha)$) for some α and some p .

First notice that $\overline{\Delta}(\mathcal{G})$ is defined as n^2 . Indeed, a vertex α of \mathcal{G} is adjacent to a vertex α' if there exists a directed path, from a vertex x_i to a vertex x_j , in some orientation $D \in \mathcal{O}_\alpha(G)$ that is directed in the other sense in some orientation $D' \in \mathcal{O}_{\alpha'}(G)$. There can be n^2 such pairs of vertices of G .

Let us now estimate t_{Adj} . Orientation D being k -arc-connected, we know by Lemma 76 that checking if a pair (x_i, x_j) is flippable can be done in time $O(km)$. If the pair of vertices is flippable, we construct orientation D' from D such that the indegree increases by 1 on vertex x_i and decreases by 1 on vertex x_j , this takes a constant time. Hence, the time to calculate t_{Adj} is in $O(km)$.

Calculating $f(\alpha)$ for some α can be done by going through all n^2 possible neighbours of α during the construction of t_{Adj} and identifying the one with smallest distance to the sink α_0 . Calculating the distance $d(\alpha, \alpha_0)$ takes a time in $O(n)$, thus, the total time to calculate t_f is in $O(n^3 \times km)$.

We can conclude that the total time of Reverse Search is

$$\begin{aligned} & O(|\mathcal{V}| \cdot \overline{\Delta}(\mathcal{G}) \cdot (t_{\text{Adj}} + t_f)) \\ &= O(|\mathcal{V}| \cdot n^2 \cdot (km + n^2 km)). \end{aligned}$$

Meanwhile, the delay is bounded not according to the total number of solutions but depending on h , the height of the tree covering \mathcal{G} described by f . The height of the tree is bounded by $\max\{d(\alpha, \alpha_0) \mid \alpha \in \mathcal{V}\}$. Since $d(\alpha, \alpha_0) = \sum_{i=1}^n |\alpha_i - \alpha_{0_i}|$ and $\sum_{i=1}^n \alpha_i = \sum_{i=1}^n \alpha_{0_i} = m$, we have $\sum_{i=1}^n |\alpha_i - \alpha_{0_i}| \leq \sum_{i=1}^n \alpha_i + \sum_{i=1}^n \alpha_{0_i} \leq 2m$ and hence we can conclude that our delay is

$$\begin{aligned} & O(h \cdot \overline{\Delta}(\mathcal{G}) \cdot (t_{\text{Adj}} + t_f)) \\ &= O(m \cdot n^2 \cdot (km + n^3 km)) \\ &= O(n^5 km^2). \end{aligned}$$

This complexity analysis can probably be improved with a more involved analysis, but we will here focus on another way of solving `ENUM_k-CONNECTED-OUTDEGREE-SEQUENCES` that will prove to be much more simple and efficient.

4.5.2 Flashlight backtrack search via path flips

As for ENUM_ α -ORIENTATIONS, we can use the flashlight backtrack search method for solving ENUM_ k -CONNECTED-OUTDEGREE-SEQUENCES. The algorithm takes as input a first arbitrary k -arc-connected orientation D with outdegree sequence α and a subset of vertices F . Then D is modified step by step into a new k -arc-connected orientation D' whose outdegree sequence differs from D . At each recursive call, a vertex v is considered, there are three possible cases to deal with:

- the k -connected outdegree sequences β with $\beta(v) = \alpha(v)$: this branch is known to be non-empty since α is a possible extension, thus we recurse with D and $F \cup v$.
- the k -connected outdegree sequences β with $\beta(v) < \alpha(v)$: we know by Lemma 58, that there exists a k -arc-connected orientation with outdegree sequence β only if there exists a vertex u such that (v, u) is flippable. If the pair (v, u) is flippable, there are two cases to consider:
 - the k -connected outdegree sequences β with $\beta(v) = \alpha(v) - 1$: we recurse with the new k -arc-connected orientation $D^{P_{vu}}$ and $F \cup v$;
 - the k -connected outdegree sequences β with $\beta(v) < \alpha(v) - 1$: we pursue the decreasing of $\alpha(v)$ applying Lemma 58 as long as possible;
- the k -connected outdegree sequences β with $\beta(v) > \alpha(v)$: we know by Lemma 58, that there exists a k -arc-connected orientation with outdegree sequence β only if there exists a vertex u such that (u, v) is flippable. If such u exists, there are two cases to consider:
 - the k -connected outdegree sequences β with $\beta(v) = \alpha(v) + 1$: we recurse with the new k -arc-connected orientation $D^{P_{uv}}$ and $F \cup v$;
 - the k -connected outdegree sequences β with $\beta(v) > \alpha(v) + 1$: we pursue the increasing of $\alpha(v)$ applying Lemma 58 as long as possible.

Algorithm 21: Enumeration of k -connected outdegree sequences**Input:** a graph $G = (V, E)$, an integer k **Output:** all k -connected outdegree sequences of G

```

1 begin
2   if there exists a  $k$ -connected orientation  $D$  for  $G$  then
3     Fix an arbitrary linear order on  $V$ ;
4     EnODS( $D, \emptyset$ );
5 end

6 Function EnODS( $D, F$ ):
7   if  $F \neq V$  then
8     take the smallest  $v \in V \setminus F$ ;
9     Reverse-( $D, F, v$ );
10    Reverse+( $D, F, v$ );
11    EnODS( $D, F \cup \{v\}$ );
12  else
13    Output  $\delta_D^+$ ;
14 end

15 Function Reverse-( $D, F, v$ ):
16   if there exists  $u \in V \setminus F$  such that  $(v, u)$  is flippable then
17     Take a directed path  $P_{vu}$  from  $v$  to  $u$  ;
18     Reverse-( $D^{P_{vu}}, F, v$ ) ;
19     EnODS( $D^{P_{vu}}, F \cup \{v\}$ );
20 end

21 Function Reverse+( $D, F, v$ ):
22   if there exists  $u \in V \setminus F$  such that  $(u, v)$  is flippable then
23     Take a directed path  $P_{uv}$  from  $u$  to  $v$  ;
24     Reverse+( $D^{P_{uv}}, F, v$ ) ;
25     EnODS( $D^{P_{uv}}, F \cup \{v\}$ );
26 end

```

Theorem 78. Let D be a k -connected orientation of $G = (V, E)$ and $F \subseteq V$. The function $\text{EnODS}(D, F)$ enumerates the k -connected outdegree sequences coinciding with D on F . Furthermore, Algorithm 21 enumerates all k -connected outdegree sequences of G exactly once and with a delay in $O(knm^2)$.

Proof. We prove the completeness of the described algorithm, then we show that there is no redundancy and finally we analyse its complexity.

Completeness We show the first part of the proposition by induction on $|V \setminus F|$. If $|V \setminus F| = 0$, then $\text{EnODS}(D, F)$ outputs δ_D^+ and the proposition holds. Consider now the case $|V \setminus F| > 0$ and let $v \in V \setminus F$ be the next vertex. By induction $\text{EnODS}(D, F \cup \{v\})$ enumerates every k -connected outdegree sequence coinciding with D on $F \cup \{v\}$ exactly once. We have to show that $\text{Reverse}^+(D, F, v)$ (resp. $\text{Reverse}^-(D, F, v)$) enumerates all k -connected outdegree sequences coinciding with D on F and having outdegree of v larger (resp. smaller) than $\delta_D^+(v)$. Also, we have to show that each of these outdegree

sequences will be enumerated exactly once. Note that this implies that globally each solution is produced exactly once.

Let us prove this for $\text{Reverse}^+(D, F, v)$. So let D' be a k -connected orientation of G such that $\delta_D^+(F) \equiv \delta_{D'}^+(F)$ and $\delta_D^+(v) < \delta_{D'}^+(v)$. By Lemma 58 item (1) there exists a vertex $u \in V \setminus F$ such that (u, v) is flippable, i.e., for any path P_{uv} the orientation $D^{P_{uv}}$ is k -connected, its outdegree sequence coincides with D on F and $\delta_D^+(v) + 1 = \delta_{D^{P_{uv}}}^+(v)$.

We proceed by induction on $\delta_{D'}^+(v) - \delta_D^+(v)$ to show that $\delta^+(D')$ is enumerated exactly once. So for the base case $\delta_{D'}^+(v) - \delta_D^+(v) = 1$ we have $\delta_{D'}^+(v) = \delta_{D^{P_{uv}}}^+(v)$ and by induction $\delta_{D'}^+$ will be enumerated exactly once by the next call of $\text{EnODS}(D^{P_{uv}}, F \cup \{v\})$ and not at all by $\text{Reverse}^+(D^{P_{uv}}, F, v)$ since the latter outputs degree sequences with $\alpha(v) > \delta_{D^{P_{uv}}}^+(v)$. Suppose now that $\delta_{D'}^+(v) - \delta_D^+(v) > 1$. We have $\delta_{D'}^+(v) - \delta_{D^{P_{uv}}}^+(v) < \delta_{D'}^+(v) - \delta_D^+(v)$, so by induction hypothesis $\text{Reverse}^+(D^{P_{uv}}, F, v)$ enumerates the outdegree sequence of $\delta_{D'}^+(v)$ exactly once.

The analogue proof works for Reverse^- using Lemma 58 item (2).

Irredundancy Each node built by this algorithm gives rise to branches that are disjoint. Function EnODS gives place to nodes with three children, one where we decrease the outdegree of the considered vertex, one where we fix it and one where we increase it. Function Reverse^- (resp. Reverse^+) generates nodes with only two sons, one where we keep trying to decrease (resp. increase) the outdegree of the considered vertex and one where we fix it.

The Figures 4.14 and 4.15 illustrate the recursion tree built by Algorithm 21. In Figure 4.14, each node is labeled with the the current test of flippability and the way D is modified. Figure 4.15 illustrates the way the outdegree sequence of the k -connected orientation D taken as input is modified into a new k -connected outdegree sequence. The coordinates in red are the ones that are fixed.

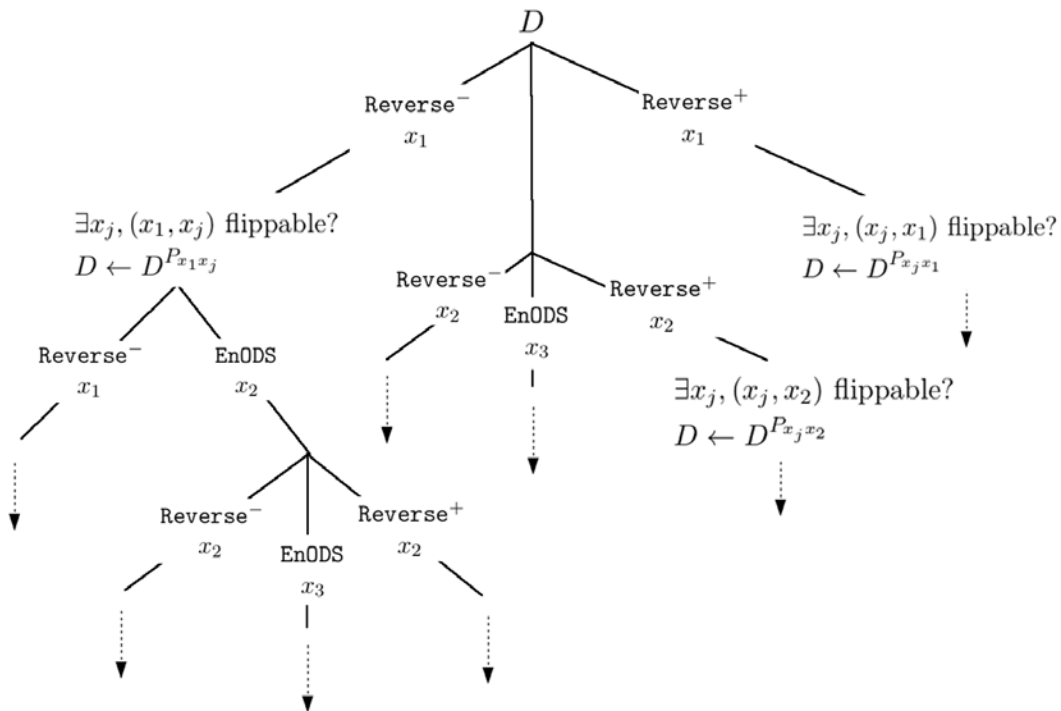
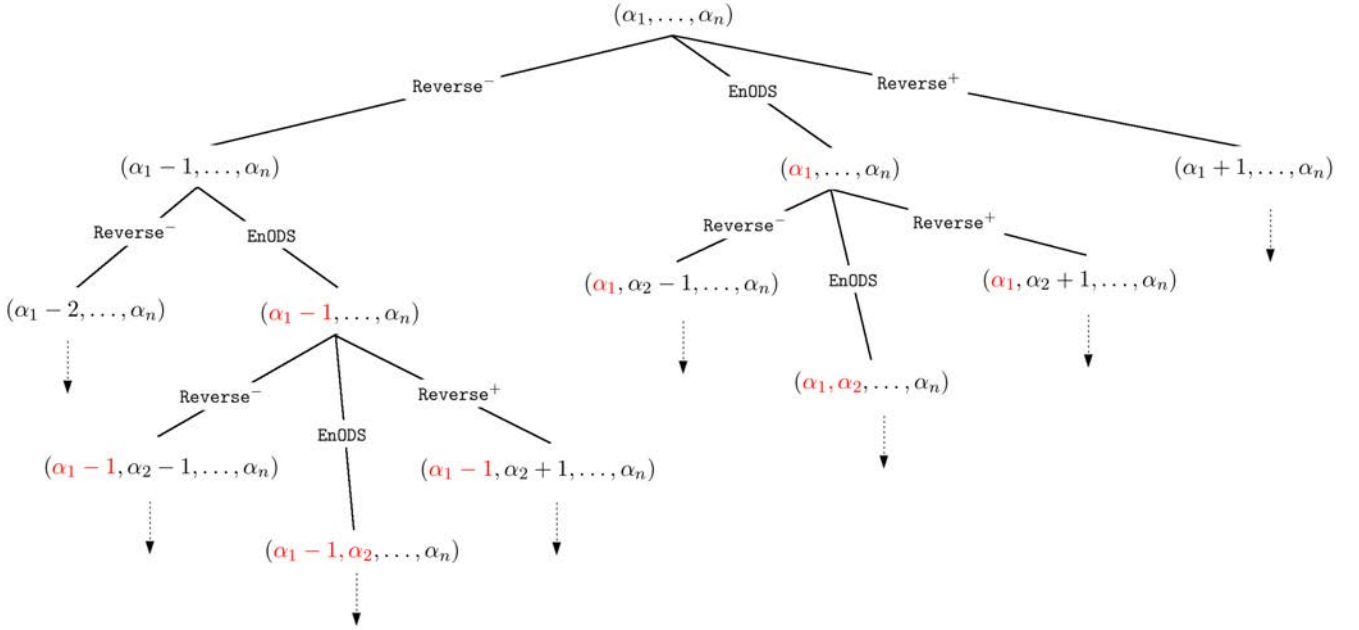


Figure 4.14: Recursion tree (1) of Algorithm 21


 Figure 4.15: Recursion tree (2) of Algorithm 21 with $\delta_D^+ = (\alpha_1, \dots, \alpha_n)$

Hence, our algorithm does not generate twice the same solution.

Complexity In each call of Reverse^+ or Reverse^- a pair (u, v) is checked to be flippable at most n times. Each check can be done in time $O(km)$ by Lemma 76 and finding a directed path from u to v is done in $O(m)$. So a call costs $O(knm)$. Finally, the depth of the recursion tree is in $O(m)$. To see this compare the $\delta_{D'}^+$ of a leaf orientation with the δ_D^+ of orientation D at the root. Between any two calls of EnODS , there will be a sequence of at most $\deg(v)$ calls of Reverse^+ or Reverse^- . This way δ_D^+ will be approached to $\delta_{D'}^+$ coordinate by coordinate, where previous coordinates are not affected by modifications on latter coordinates. Thus, there are at most $\sum_{v \in V} \deg(v) = O(m)$ calls.

Note that Algorithm 21 has to use a separate method for finding a first k -connected orientation D of G . This pre-processing step can be done in $O(k^3n^3 + kn^2m)$ [62]. Recall that in a k -connected orientation we have $kn \leq m$. Therefore, we get an overall delay of $O(knm^2)$. \square

4.6 Simple and modular enumeration of k -connected orientations

In the two previous sections, we have solved $\text{ENUM_}\alpha\text{-ORIENTATIONS}$ and $\text{ENUM_}k\text{-CONNECTED-OUTDEGREE-SEQUENCES}$ each time using two different methods:

1. $\text{ENUM_}\alpha\text{-ORIENTATIONS}$
 - (a) Flashlight backtrack search via integer points of polytopes
 - (b) Flashlight backtrack search via cycle flips
2. $\text{ENUM_}k\text{-CONNECTED-OUTDEGREE-SEQUENCES}$
 - (c) Reverse search
 - (d) Flashlight backtrack search via path flips

The best complexities we derived are for (b) and (d). Combining algorithms 19 and 21 we obtain a simple, modular algorithm to enumerate k -connected orientations. The fact that it is modular allows us to give a better analysis of the amortized time, that improves over what could be expected from simple EXT SOL approaches as outlined in Section 4.3.

Final Algorithm 22 uses Algorithm 21 for finding all k -connected outdegree sequences. Each time a new outdegree sequences is found, Algorithm 19 is called in order to generate all orientations respecting this outdegree sequence.

Algorithm 22: Simple enumeration of k -connected orientations

Input: a graph $G = (V, E)$, an integer k
Output: all k -connected orientations of G

```

1 begin
2   if there exists a  $k$ -connected orientation  $D$  for  $G$  then
3     Fix an arbitrary linear order on  $V$ ;
4     EnODS'( $D, \emptyset$ );
5 end

6 Function EnODS'( $D, F$ ):
7   if  $F \neq V$  then
8     take the smallest  $v \in V \setminus F$ ;
9     Reverse-( $D, F, v$ );
10    Reverse+( $D, F, v$ );
11    EnODS'( $D, F \cup \{v\}$ );
12  else
13    EnOPODS( $D, \emptyset$ );
14 end

```

See the Figure 4.16 for an illustration of the recursion tree built by final Algorithm 22.

We need the following easy result for analysing the amortized complexity.

Lemma 79. *Let $G = (V, E)$ be a graph and α a k -connected outdegree sequence, then $|\mathcal{O}_\alpha(G)| \geq (k - 1)n + 2$.*

Proof. Let $D \in \mathcal{O}_\alpha(G)$. We will show that D contains at least $(k - 1)n + 1$ directed cycles. Since for each directed cycle C , the orientation D^C is a different element of $\mathcal{O}_\alpha(G)$, we obtain the result.

Let $G = (V, E)$ be an undirected graph and \mathcal{C} be the set of all cycles of G . Let $D \in \mathcal{O}_\alpha(G)$. We associate to \mathcal{C} and D a vector space \mathcal{X} as follows. We give a direction of traversal for each cycle C of \mathcal{C} and we associate to each such cycle a vector $x_C \in \mathbb{R}^m$:

$$\forall a \in D : x_C(a) = \begin{cases} 1 & \text{if } a \in C \\ -1 & \text{if } a^- \in C \\ 0 & \text{else.} \end{cases}$$

It is well known, that in the case of strongly connected graphs, for such a vector space there exists a base constituted of the x_C where C is directed in D , see [50]. Moreover, it can be found in most books on

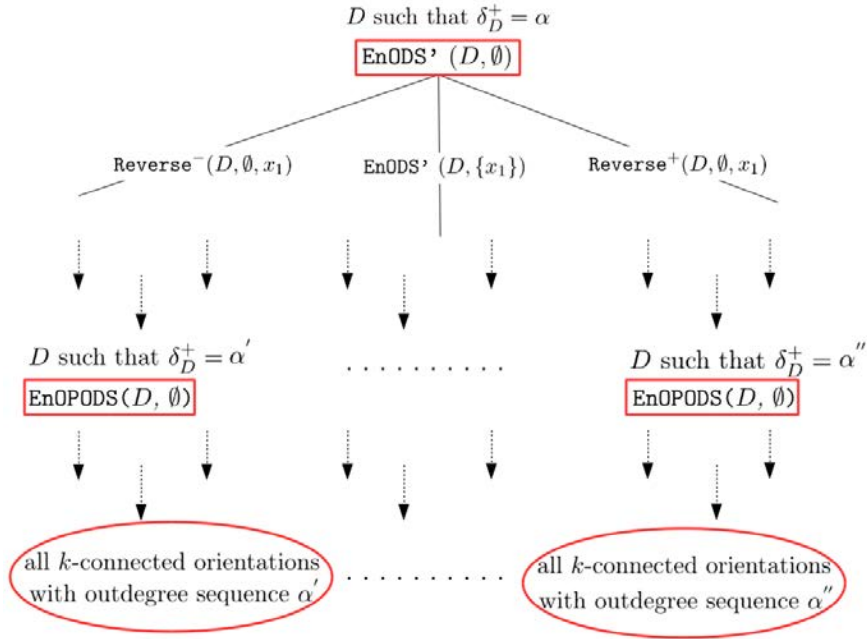


Figure 4.16: Algorithm 22 with $G = (V, E)$ and $V = \{x_1, \dots, x_n\}$

(algebraic) graph theory that in the case of weakly connected digraphs, the dimension of such a vector space is $m - n + 1$, see e.g. [51, 71]. Hence, the number of directed cycles of D is at least $m - n + 1$. Orientation D being k -arc-connected, $m \geq kn$ and thus D contains at least $(k - 1)n + 1$ directed cycles. \square

Theorem 80. *Let G be a graph and $k \in \mathbb{N}$. Algorithm 22 enumerates all k -connected orientations of G with delay $O(knm^2)$. If $k \geq 2$ the amortized time is in $O(m^2)$.*

Proof. Recall that the pre-processing step, which consists in finding a first k -connected orientation, takes a time in $O(k^3n^3 + kn^2m)$ [62]. Thus, the correctness and the delay follow directly from Theorems 71 and 78, see Figure 4.17.

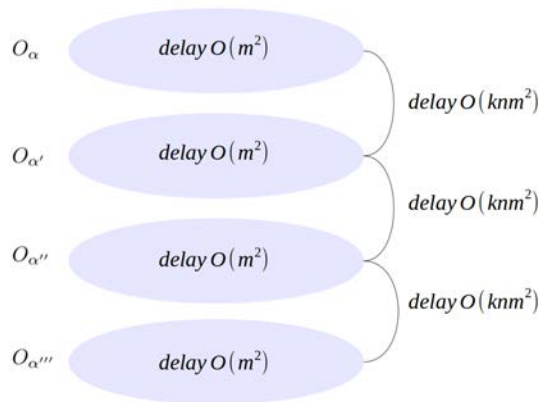


Figure 4.17: Delay of final Algorithm 22

Let us compute the amortized time complexity as an average over the delays. Let s be the number of solutions, i.e., the total number of k -connected orientations and t be the number of k -connected outdegree sequences of G . There exists some constants c and c' , such that the overall running time of our algorithm is bounded by $cknm^2t + c'm^2s$. Since for every k -connected outdegree sequence α there are at least $(k-1)n + 1$ orientations, we have that $t \leq \frac{s}{(k-1)n+1}$ and thus

$$cknm^2t + c'm^2s \leq cknm^2 \frac{s}{(k-1)n} + c'm^2s = O(m^2)s$$

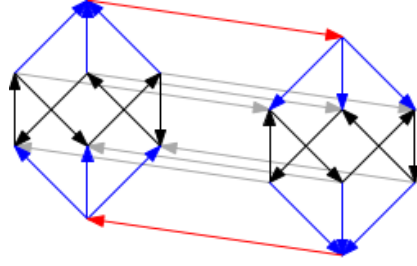
where for the last equality we use $k \geq 2$. Hence the amortized complexity is in $O(m^2)$. □

Remark 7. Note that this analysis requires $k > 1$. The case $k = 1$ is solved in [21] where an enumeration algorithm with delay $O(m)$ is presented for strongly connected orientations. Hence, we can conclude that the problem of generating k -arc-connected orientation can be solved in amortized time $O(m^2)$ for all k .

4.7 Conclusion

As seen in Section 4.3, the fact that we can decide in polynomial time if a mixed graph can be extended to a k -connected orientation, leads to Algorithm 14 which enumerates all k -connected orientations with delay in $O(m(k^3n^3 + kn^2m))$. However, understanding and implementing the extension of a mixed graph to a k -arc-connected one, is rather intricate. We proposed a careful analysis of the problem ENUM_ k -ARC-CONNECTED-ORIENTATIONS by dividing it into two subproblems of independent interest. We proposed for the first one - enumeration of all orientations respecting some fixed outdegree sequence - two different ways of solving it: through a flashlight backtrack search and the reduction to the problem of enumerating the vertices of a 0/1-polytope, and through a flashlight backtrack search algorithm via cycle flips. We cast the first problem as a special case of integer point enumeration of box-integer polytopes, for which we gave a simple polynomial delay algorithm. For the second subproblem - generation of all k -connected outdegree sequences - we proposed again two different ways of solving it: through a reverse search and through a flashlight backtrack search.

For the generation of all α -orientations we thought of using a similar reverse search as presented in Section 4.5.1 for the enumeration of k -connected outdegree sequences: the metagraph has as vertex set the α -orientations and there exists an edge between two α -orientations D and D' if D' can be obtained from D by the reversal of one directed cycle. This technique works only in the case where there exists a subset of cycles X of the graph G such that X is polynomially bounded in $|G|$ and any directed cycle of G can be obtained by reversing a sequence of directed cycles of X . In [34] a notion of *essential cycles* in the case of planar graphs is introduced. It is shown that the number of these essential cycles is linear in the number of vertices of G and that with reorientations of essential cycles we can commute between any two α -orientations. Hence, such reverse search algorithm would work for planar graphs. Using the same idea as in [34], we did also convince ourselves that such cycle sets exist for graphs of bounded genus and k -chordal graphs. However, for general graphs, the existence of such a set is open. A first open graph class are hypercubes. In particular, it is open whether there is a constant c such that every strong orientation contains a directed cycle of length at most c . With the example of Figure 4.18, shown to us by Torsten Ueckerdt, we know that $c > 4$.

Figure 4.18: A strong orientation of Q_4 with no directed cycle of length at most 4

For both subproblems $\text{ENUM_}\alpha\text{-ORIENTATIONS}$ and $\text{ENUM_}k\text{-CONNECTED-OUTDEGREE-SEQUENCES}$, the best complexities we derived came from the second method. Together these two flashlight backtrack search algorithms gave us an alternative solution, Algorithm 22, for the enumeration of k -connected orientations. This resulting algorithm is very simple, has a better delay and even better amortized running time than Algorithm 14, stated in Section 4.3. Table 4.7 summarizes the different methods presented in the chapter to solve $\text{ENUM_}k\text{-ARC-CONNECTED-ORIENTATIONS}$.

Problem	Algorithm	Delay
$\text{ENUM_}\alpha\text{-ORIENTATIONS}$	Reduction to $\text{ENUM_VERTICES-0/1-POLYTOPE}$ Backtrack search and cycle flips (1)	$O(d(l, u)(d^8 \cdot \log^2 T))$ $O(m^2)$
$\text{ENUM_}k\text{-CONNECTED-OUTDEGREE-SEQUENCES}$	Reverse Search Backtrack search and flippable paths (2)	$O(n^5 km^2)$ $O(knm^2)$
$\text{ENUM_}k\text{-ARC-CONNECTED-ORIENTATIONS}$	Backtrack search and extension of partial orientation Backtrack search (1) + (2)	$O(m(k^3 n^3 + kn^2 m))$ delay: $O(knm^2)$ amortized: $O(m^2)$

Table 4.1: Recap table for $\text{ENUM_}k\text{-ARC-CONNECTED-ORIENTATIONS}$

Maybe the delay of our method could be improved with the introduction of additional sophisticated data structures. For example it is maybe possible to decrease the complexity of the BFS during the generation of the α -orientations. In Algorithm 19, a BFS is used each time to check the existence of a path in a graph that differs from the previous graph only by the direction of one cycle. One could expect that having this information the BFS running time could be improved. It may also be possible to decrease the complexity of the test of flippability during the generation of the k -connected outdegree sequences. Algorithm 21 uses as subroutine Algorithm 20 which verifies if a pair of vertices is flippable by reversing k paths between these vertices. Maybe we can reduce this number of reversals with particular data structures. About the amortized time, the complexity of $O(m^2)$ is a particularly interesting result. In fact, to get such a good time with Algorithm 14, we would need to be able to do partial orientation extension in $O(m)$, which is far better than the best-known algorithms.

The weakness of our approach is that finding the initial solution, i.e., finding a k -connected orientation of a graph, is algorithmically the most complicated part. We already mentioned the complexity $O(n^3 k^3 +$

kn^2m) by Iwata and Kobayashi in [62] and Lovász's result which yields a running time in $O(n^6)$. The best improvements and implementations using splitting off techniques lead to computation time of roughly $O(n^5)$ [44, 74], it would be interesting to find simpler methods.

A dual analogue of k -arc-connectivity could be called k -acyclicity, where at least k arcs of D have to be contracted in order to destroy its acyclicity. As mentioned in the introduction, acyclic orientations can be enumerated with polynomial delay. On the other hand, it is easy to see that a graph G admits a 2-acyclic orientation if and only if G is the cover graph of a poset. The corresponding recognition problem is NP-complete [16] and the proof can be extended to show that testing whether G admits a k -acyclic orientation is NP-complete for any $k \geq 2$.

Using [26, Theorem 13] one can show that enumerating k -acyclic orientations (as well as k -vertex connected orientations for $k > 2$) are DelNP-hard under D-reductions and IncNP-hard under I-reduction. Since with an NP-oracle one can decide if a given partial orientation extends to one of these types, a flashlight backtrack search algorithm like Algorithm 14 yields that the above problems can be solved with a polynomial number of calls of an NP-oracle. Thus these problems are in the class DelNP (and *a fortiori* IncNP), and therefore are indeed complete in both settings.

Another notion very close to k -arc-connectivity is the k -vertex-connectivity. An orientation of a graph is k -vertex-connected if it is strongly connected and if the removal of at least k vertices is needed to destroy the strong connectivity. Checking if a graph admits such an orientation has been proven to be polynomial for $k \leq 2$ but NP-hard for $k > 2$ [28]. Little is known about vertex-connected orientations. It would be interesting to make a similar work for k -vertex-connectivity as the one done in Chapter 4 for k -arc-connectivity in order to answer the question whether such orientations can be enumerated in polynomial delay as well. In [5] Bang-Jensen *et al.* pose as an open problem the question if a partial orientation can be extended into a 2-vertex connected orientation in polynomial time. A positive answer to this question would yield a polynomial delay enumeration algorithm by the EXTSOL method, as we did for k -arc-connected orientations in Section 4.3.

Conclusion

We addressed two issues from graph theory: the enumeration of vertex set properties and the enumeration of k -arc-connected orientations. For these two problems, we developed efficient output-sensitive algorithms.

In Chapter 3, we carried out a general study of the former. We proposed a general framework that allows for the study of enumeration of a large class of vertex set properties in graphs. We proved that when such a property is locally definable with respect to some linear order on the vertices, the corresponding enumeration problem reduces to the enumeration of paths in directed acyclic graphs. This provides a general method to design linear delay algorithms for the enumeration of such vertex set properties. We applied this general method to enumerate minimal (connected) dominating sets and maximal irredundant sets in interval graphs and in permutation graphs with linear delay. We extended this framework to cyclically ordered graphs and thus obtained linear delay algorithms for generating minimal dominating sets and maximal irredundant sets in circular-arc graphs and in circular-permutation graphs.

The collection of specific problems covered by this technique is various, but it also leaves room for further research. An obvious extension of this work would be *extending the scope of the method* by searching for additional vertex set properties and classes of graphs to which the method can apply. From a methodological point of view, an interesting line of research could be *identifying key problems*, like the enumeration of paths in a DAG, that help designing efficient enumeration algorithms through reductions to them.

The second part of the thesis was devoted to the study of the enumeration of k -arc-connected orientations. In Chapter 4, we proposed a simple and polynomial delay algorithm to solve this task. Our algorithm has quadratic amortized time, which seems hard to improve with current techniques.

The construction of this enumeration algorithm gave rise to the study of two different enumeration problems of independent interest: the enumeration of orientations respecting some fixed outdegree vector and the enumeration of outdegree sequences for which there exists at least one k -arc-connected orientation. The weakness of our approach lays in the pre-processing, with the search for a first k -connected orientation of a graph. This task is algorithmically the most complicated and time-consuming part of our algorithm and further study should be conducted to find a simpler pre-processing method.

Another natural way of pursuing this work is to find efficient enumeration algorithms for further sets of orientations or prove the hardness of these problems. We already discussed k -vertex-connected and k -acyclic orientations. Another natural class would be that of (k -arc-connected) orientations of given digirth d , i.e. orientations in which all directed cycles are of length at least d .

Furthermore, a more general direction of research is about the comparison of counting complexity and enumeration complexity. It appears that, as far as graph orientation is concerned, there are more results and research about counting than about enumeration. Is there a generic framework that links

enumeration algorithms and counting algorithms for orientations? A particularly interesting line of work is suggested by the Tutte polynomial of a graph G whose evaluations give the size of different sets of objects associated with G . In particular, it counts the acyclic orientations, the outdegree sequences, and the strongly connected orientations of G , see [8, 49, 70]. Can the recursive definition of the Tutte polynomial be used to get efficient enumeration algorithms for the counted sets of objects? In particular what would be the delay for enumerating outdegree sequences (not necessarily k -connected)?

Several contributions to enumeration theory are made in this thesis, through the development of both general methods and specific algorithms. In particular, the very general study of vertex set properties gives an insight into the structure that allows for linear delay enumeration for a broad class of problems. General results of this form, which allow the design of algorithms for a whole class of problems, are rare in the field of algorithmic enumeration. In contrast, our study of k -arc-connected orientations examines different ways to tackle some specific enumeration problems. It completes a detailed comparison of different enumerative approaches for these problems.

The enumeration complexity is a rather recent issue, its main concepts and definitions just begin to stabilize. In particular, the two fields developed in this thesis contain a lot of open questions and in view of the technological developments, application domains will become more numerous. Thus, it leaves room for a large field of inquiries for further research.

Bibliography

- [1] O. Aichholzer, J. Cardinal, T. Huynh, K. Knauer, T. Mütze, R. Steiner, and B. Vogtenhuber. Flip distances between graph orientations. In *Graph-Theoretic Concepts in Computer Science - 45th International Workshop, WG 2019, Vall de Núria, Spain, June 19-21, 2019, Revised Papers*, pages 120–134, 2019.
- [2] T. Aittokallio and B. Schwikowski. Graph-based methods for analysing networks in cell biology. *Briefings in Bioinformatics*, 7(3):243–255, 2006.
- [3] D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1-3):21–46, 1996.
- [4] G. Bagan, A. Durand, and E. Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings*, pages 208–222, 2007.
- [5] J. Bang-Jensen, J. Huang, and X. Zhu. Completing orientations of partially oriented graphs. *Journal of Graph Theory*, 87(3):285–304, 2018.
- [6] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *Journal of the ACM*, 48(5):1069–1090, 2001.
- [7] V. C. Barbosa and J. L. Szwarcfiter. Generating all the acyclic orientations of an undirected graph. *Information Processing Letters*, 72(1):71 – 74, 1999.
- [8] O. Bernardi. Tutte polynomial, subgraphs, orientations and sandpile model: New connections via embeddings. *Electronic Journal of Combinatorics*, 15(1), 2008.
- [9] H. L. Bodlaender, T. Kloks, and D. Kratsch. Treewidth and pathwidth of permutation graphs. *SIAM Journal on Discrete Mathematics*, 8(4):606–616, 1995.
- [10] F. Boesch and R. Tindell. Robbins’ theorem for mixed multigraphs. *The American Mathematical Monthly*, 87(9):716–719, 1980.
- [11] M. Bonamy, O. Defrain, M. Heinrich, and J.-F. Raymond. Enumerating minimal dominating sets in triangle-free graphs. In *STACS 2019, March 13-16, 2019, Berlin, Germany*, pages 16:1–16:12, 2019.
- [12] A. Bondy and U.S.R. Murty. In *Graph Theory. Graduate Texts in Mathematics*. Springer, 2008.
- [13] E. Boros, V. Gurvich, L. Khachiyan, and K. Makino. Generating partial and multiple transversals of a hypergraph. In *Automata, Languages and Programming, 27th International Colloquium, ICALP 2000, Geneva, Switzerland, July 9-15, 2000, Proceedings*, pages 588–599, 2000.
- [14] E. Boros and K. Makino. 2016. Personal communication.

- [15] E. Brehm. 3-orientations and Schnyder-3-tree-decompositions, 2000. Diploma Thesis, FU Berlin.
- [16] G. Brightwell. On the complexity of diagram testing. *Order*, 10(4):297–303, Dec 1993.
- [17] M. R. Bussieck and M. E. Lübbecke. The vertex set of a 0/1-polytope is strongly p-enumerable. *Computational Geometry*, 11(2):103–109, 1998.
- [18] F. R. K. Chung, M. R. Garey, and R. E. Tarjan. Strongly connected orientations of mixed multi-graphs. *Networks*, 15(4):477–484, 1985.
- [19] A. Conte. *Enumeration Algorithms for Real-World Networks: Efficiency and Beyond*. PhD thesis, Università di Pisa, 2018.
- [20] A. Conte, R. Grossi, A. Marino, and R. Rizzi. Efficient enumeration of graph orientations with sources. *Discrete Applied Mathematics*, 246:22–37, 2018.
- [21] A. Conte, R. Grossi, A. Marino, R. Rizzi, and L. Versari. *Directing Road Networks by Listing Strong Orientations*, pages 83–95. Springer International Publishing, 2016.
- [22] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. The MIT Press and McGraw-Hill Book Company, 2001.
- [23] J.-F. Couturier, P. Heggernes, P. van’t Hof, and D. Kratsch. Minimal dominating sets in graph classes: Combinatorial bounds and enumeration. *Theoretical Computer Science*, 487:82–94, 2013.
- [24] J.-F. Couturier, R. Letourneur, and M. Liedloff. On the number of minimal dominating sets on some graph classes. *Theoretical Computer Science*, 562:634–642, 2015.
- [25] N. Creignou and J.-J. Hébrard. On generating all solutions of generalized satisfiability problems. *Informatique Théorique et Applications*, 31(6):499–511, 1997.
- [26] N. Creignou, M. Kröll, R. Pichler, S. Skritek, and H. Vollmer. On the complexity of hard enumeration problems. In *Language and Automata Theory and Applications - 11th International Conference, LATA 2017, Umeå, Sweden, March 6-9, 2017, Proceedings*, pages 183–195, 2017.
- [27] W. H. Cunningham and A. Frank. A primal-dual algorithm for submodular flows. *Mathematics of Operations Research*, 10(2):251–262, 1985.
- [28] O. D. de Gevigney. On frank’s conjecture on k -connected orientations. *CoRR*, abs/1212.4086, 2012.
- [29] A. Durand and E. Grandjean. First-order queries on structures of bounded degree are computable with constant delay. *ACM Transactions on Computational Logic*, 8(4):21, 2007.
- [30] G. Ehrlich. Loopless algorithms for generating permutations, combinations, and other combinatorial configurations. *Journal of the ACM*, 20(3):500–513, 1973.
- [31] T. Eiter and G. Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing*, 24(6):1278–1304, 1995.
- [32] S. Even, A. Pnueli, and A. Lempel. Permutation graphs and transitive graphs. *Journal of the ACM*, 19(3):400–410, 1972.
- [33] M. Fellows, G. Fricke, S. Hedetniemi, and D. Jacobs. The private neighbor cube. *SIAM Journal on Discrete Mathematics*, 7:41–47, 1994.

-
- [34] S. Felsner. Lattice structures from planar graphs. *Electronic Journal of Combinatorics*, 11(1), 2004. Research Paper 15.
- [35] S. Felsner. Rectangle and square representations of planar graphs. In *Thirty essays on geometric graph theory*, pages 213–248. Springer, New York, 2013.
- [36] S. Felsner, H. Schrezenmaier, and R. Steiner. Equiangular polygon contact representations. In *Graph-Theoretic Concepts in Computer Science - 44th International Workshop, WG 2018, Cottbus, Germany, June 27-29, 2018, Proceedings*, pages 203–215, 2018.
- [37] S. Felsner, H. Schrezenmaier, and R. Steiner. Pentagon contact representations. *The Electronic Journal of Combinatorics*, 25(3), 2018. Paper 3.39, 38.
- [38] F. V. Fomin, F. Grandoni, A. V. Pyatkin, and A. A. Stepanov. Combinatorial bounds via measure and conquer: Bounding minimal dominating sets and applications. *ACM Transactions on Algorithms*, 5(1):9:1–9:17, 2008.
- [39] A. Frank. An algorithm for submodular functions on graphs. *Annals of Discrete Mathematics*, 16:97–120, 1982.
- [40] A. Frank. A note on k -strongly connected orientations of an undirected graph. *Discrete Mathematics*, 39(1):103–104, 1982.
- [41] A. Frank. Submodular flows. *Progress in Combinatorial Optimization*, pages 147–165, 1984.
- [42] A. Frank. Applications of submodular functions. In *Surveys in combinatorics, 1993. Papers of the fourteenth British combinatorial conference held at Keele, United Kingdom, July 1993*, pages 85–136. Cambridge: Cambridge University Press, 1993.
- [43] H. N. Gabow. A framework for cost-scaling algorithms for submodular flow problems. In *34th Annual Symposium on Foundations of Computer Science, Palo Alto, California, USA, 3-5 November 1993*, pages 449–458, 1993.
- [44] H. N. Gabow. Efficient splitting off algorithms for graphs. In *Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing, STOC '94*, pages 696–705, New York, NY, USA, 1994. ACM.
- [45] H. N. Gabow. Centroids, representations, and submodular flows. *Journal of Algorithms*, 18(3):586–628, 1995.
- [46] P. Gara and R. Frank. Gray codes from antimatroids. *Order*, 10(3):239–252, 1993.
- [47] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman, 1979.
- [48] P. M. Gilmer and R. A. Litherland. The duality conjecture in formal knot theory. *Osaka Journal of Mathematics*, 23(1):229–247, 1986.
- [49] E. Gioan. Enumerating degree sequences in digraphs and a cycle-cocycle reversing system. *European Journal of Combinatorics*, 28(4):1351–1366, 2007.
- [50] P. M. Gleiss, J. Leydold, P. C. Godsil, G. Royle, and F. Stadler. Circuit bases of strongly connected digraphs. *Discussiones Mathematicae, Graph Theory*, 23(2):241–260, 2003.

- [51] C. Godsil and G. Royle. *Algebraic graph theory*, volume 207. New York, NY: Springer, 2001.
- [52] P. A. Golovach, P. Heggernes, M. M. Kanté, D. Kratsch, S. H. Sæther, and Y. Villanger. Output-polynomial enumeration on graphs of bounded (local) linear mim-width. *Algorithmica*, 80(2):714–741, 2018.
- [53] P. A. Golovach, P. Heggernes, and D. Kratsch. Enumerating minimal connected dominating sets in graphs of bounded chordality. *Theoretical Computer Science*, 630:63–75, 2016.
- [54] P. A. Golovach, D. Kratsch, and M. Y. Sayadi. Enumeration of maximal irredundant sets for claw-free graphs. *Theoretical Computer Science*, 754:3–15, 2019.
- [55] M. C. Golumbic. Interval graphs and related topics. *Discrete Mathematics*, 55(2):113–121, 1985.
- [56] D. Gonçalves, B. Lévêque, and A. Pinlou. Triangle contact representations and duality. *Discrete and Computational Geometry. An International Journal of Mathematics and Computer Science*, 48(1):239–254, 2012.
- [57] D. Gunopulos, R. Khardon, H. Mannila, and H. Toivonen. Data mining, hypergraph transversals, and machine learning. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 12-14, 1997, Tucson, Arizona, USA*, pages 209–216, 1997.
- [58] M. Habib, R. Medina, L. Nourine, and G. Steiner. Efficient algorithms on distributive lattices. *Discrete Applied Mathematics*, 110(2):169 – 187, 2001.
- [59] S. T. Hedetniemi and R. C. Laskar. In *Topics on Domination. North-Holland*, 1991.
- [60] M. A. Henning and A. Yeo. In *Total Domination in Graphs. Springer*, 2013.
- [61] H. Hosoya. Topological index. a newly proposed quantity characterizing the topological nature of structural isomers of saturated hydrocarbons. *Bulletin of The Chemical Society of Japan - BULL CHEM SOC JPN*, 44:2332–2339, 01 1971.
- [62] S. Iwata and Y. Kobayashi. An algorithm for minimum cost arc-connectivity orientations. *Algorithmica*, 56(4):437–447, 2010.
- [63] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988.
- [64] M. M. Kanté, V. Limouzy, A. Mary, and L. Nourine. Enumeration of minimal dominating sets and variants. In *Fundamentals of Computation Theory - 18th International Symposium, FCT 2011, Oslo, Norway, August 22-25, 2011. Proceedings*, pages 298–309, 2011.
- [65] M. M. Kanté, V. Limouzy, A. Mary, and L. Nourine. On the enumeration of minimal dominating sets and related notions. *SIAM Journal on Discrete Mathematics*, 28(4):1916–1929, 2014.
- [66] M. M. Kanté, V. Limouzy, A. Mary, L. Nourine, and T. Uno. On the enumeration and counting of minimal dominating sets in interval and permutation graphs. In *Algorithms and Computation - 24th International Symposium, ISAAC 2013, Hong Kong, China, December 16-18, 2013, Proceedings*, pages 339–349, 2013.
- [67] R. M. Karp. Reducibility among combinatorial problems. In *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*, pages 219–241. 2010.

-
- [68] L. G. Khachiyan, E. Boros, K. M. Elbassioni, V. Gurvich, and K. Makino. On the complexity of some enumeration problems for matroids. *SIAM Journal on Discrete Mathematics*.
- [69] S. Klamt, U.-U. Haus, and F. J. Theis. Hypergraphs and cellular networks. *PLoS Computational Biology*, 5(5), 2009.
- [70] D. J. Kleitman and K. J. Winston. Forests and score vectors. *Combinatorica*, 1(1):49–54, 1981.
- [71] U. Knauer and K. Knauer. *Algebraic graph theory. Morphisms, monoids and matrices (to appear). 2nd revised and extended edition.*, volume 41. Berlin: De Gruyter, 2nd revised and extended edition edition, 2019.
- [72] D. E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 2: Generating All Tuples and Permutations (Art of Computer Programming)*. Addison-Wesley Professional, 2005.
- [73] P. C. B. Lam and H. Zhang. A distributive lattice on the set of perfect matchings of a plane bipartite graph. *Order*, 20(1):13–29, 2003.
- [74] L. C. Lau and C. K. Yung. Efficient edge splitting-off algorithms maintaining all-pairs edge-connectivities. *SIAM Journal on Discrete Mathematics*, 42(3):1185–1200, 2013.
- [75] L. Lovász. *Combinatorial Problems and Exercises*. North-Holland, 1979.
- [76] C. L. Lucchesi and S. L. Osborn. Candidate keys for relations. *Journal of Computer and System Sciences*, 17(2):270–279, 1978.
- [77] K. Makino and T. Uno. New algorithms for enumerating all maximal cliques. In *Algorithm Theory - SWAT 2004, 9th Scandinavian Workshop on Algorithm Theory, Humlebaek, Denmark, July 8-10, 2004, Proceedings*, pages 260–272, 2004.
- [78] A. Marino. Algorithms for biological graphs: Analysis and enumeration. *Bulletin of the EATCS*, 114, 2014.
- [79] A. Mary. *Énumération des dominants minimaux d'un graphe*. PhD thesis, Université Blaise Pascal, 2013.
- [80] A. Mary and Y. Strozecki. Efficient enumeration of solutions produced by closure operations. *Discrete Mathematics and Theoretical Computer Science*, 21(3), 2019.
- [81] N. Megiddo. On the complexity of linear programming. *Advances in Economic Theory: Fifth World Congress*, 1987.
- [82] K. Menger. Zur allgemeinen kurventheorie. In *Fundamental Mathematics*, pages 96–115, 1927.
- [83] C. S. J. A. Nash-Williams. On orientations, connectivity and odd-vertex-pairings in finite graphs. *Canadian Journal of Mathematics*, 12:555–567, 1960.
- [84] J. B. Orlin. Max flows in $O(nm)$ time, or better. In *Proceedings of the 45th annual ACM symposium on theory of computing, STOC '13. Palo Alto, CA, USA, June 1–4, 2013*, pages 765–774. 2013.
- [85] J. Propp. Lattice structure for orientations of graphs. ArXiv: math/0209005, September 2002.
- [86] E. Rémila. The lattice structure of the set of domino tilings of a polygon. *Theoretical Computer Science*, 322(2):409–422, 2004.

- [87] H. E. Robbins. A theorem on graphs, with an application to a problem of traffic control. *The American Mathematical Monthly*, 46(5):281–283, 1939.
- [88] M. Rospocher. *On the computational complexity of enumerating certificates of NP problems*. PhD thesis, Università degli Studi di Trento, 2006.
- [89] J. Schmidt. *Enumeration: Algorithms and Complexity*. PhD thesis, Université de la Méditerranée, 2009.
- [90] A. Schrijver. *Theory of linear and integer programming*. Wiley-Interscience series in Discrete Mathematics and optimization. Wiley, 1999.
- [91] A. Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency*. Springer, 2003.
- [92] N. Z. Shor and V. I. Gershivich. *Family of algorithms for solving convex programming problems*. Cybernetics and Systems Analysis, Volume 15, Issue 4, pp 502-508. Springer, 1979.
- [93] M. B. Squire. Generating the acyclic orientations of a graph. *Journal of Algorithms*, 26(2):275 – 290, 1998.
- [94] É. Tardos, C. A. Tovey, and M. A. Trick. Layered augmenting path algorithms. *Mathematics of Operations Research*, 11(2):362–370, 1986.
- [95] C. Thomassen. Strongly 2-connected orientations of graphs. *Journal of Combinatorial Theory, Series B*, 110:67 – 78, 2015.
- [96] W. P. Thurston. Conway’s tiling groups. *American Mathematical Monthly*, 97(8):757–773, 1990.
- [97] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.
- [98] P.-J. Wan and D.-Z. Du. In *Connected Dominating Set : Theory and Applications*. Springer.
- [99] J. Wu and H. Li. On calculating connected dominating set for efficient routing in ad hoc wireless networks. In *Proceedings of the 3rd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIAL-M 1999), Seattle, Washington, USA, August 20, 1999*, pages 7–14, 1999.
- [100] P. Zhang, E. A. Schon, S. G. Fischer, E. Cayanis, J. Weiss, S. Kistler, and P. E. Bourne. An algorithm based on graph theory for the assembly of contigs in physical mapping of DNA. *Computer Applications in the Biosciences*, 10(3):309–317, 1994.
- [101] G. M. Ziegler. *Lectures on polytopes*. Springer-Verlag, New York, 1995.