Complétions d'intervalles minimales

Karol Suchan

UNIVERSITE D'ORLEANS

THESE PRESENTEE A L'UNIVERSITE D'ORLEANS

POUR OBTENIR LE GRADE DE

DOCTEUR DE L'UNIVERSITE D'ORLEANS

Discipline : Informatique

PAR

M. Karol SUCHAN

Sujet : Complétions d'intervalles minimales

Soutenue le **: 12 décembre 2006**

MEMBRES DU JURY :

- Mme Anne BERRY (LIMOS, Clermont Ferrand)	**Examinateur**
- M. Krzysztof DIKS (Université de Varsovie)	**Rapporteur**
- M. Pierre FRAIGNIAUD (LRI, Université Paris XI)	**Rapporteur**
- M. Michel HABIB (LIAFA, Université Paris 7)	**Examinateur**
- M. Henri THUILLIER (LIFO, Université d'Orléans)	**Directeur de thèse**
- M. Ioan TODINCA (LIFO, Université d'Orléans)	**Encadrant**

Contents

List of Figures

Chapter 1

Introduction

Research on NP-hard problems is in the core of contemporary algorithmics. Since it is impractical to search for general exact solutions to the problems in this class, many special approaches have been developed. Most efforts can be regrouped about the following main axis: heuristics, guaranteed quality approximations, exact solutions for restrained graph classes, and the most recently, general exact exponential time algorithms. Heuristics are often fast and prove very useful in applications but admit no theoretical proofs of quality. They are evaluated in empirical manner with benchmarks running on real-life data coming from applications or on random data-sets generated from adequate probability distributions. Approximation algorithms are polynomial time algorithms that give solutions that fit within bounds proved to be close to optimum. How tight these bounds can be depends on the problem. The most common are the approximations within a constant multiplicative factor, meaning that their accuracy is independent from the instance. But sometimes multiplicative constant factor polynomial time approximations are proved not to exist (again, unless P=NP). Graph coloring is an important example of such a problem. The third approach to NP-hard problems consists in finding polynomial time exact algorithms for restrained graph classes. Here the question is how strong constraints on the structure of the graph in question we need to have in order to be able to solve the problem in polynomial time. It is the usual path to start with strong conditions sufficient to prove polynomial time complexity, that are progressively relaxed as the understanding of the problem improves. Thus the class of graphs for which the problem can be efficiently solved becomes larger and larger. Eventually, the study may accomplish by finding conditions that are also necessary, hence a full characterization of instances solvable in polynomial time is given. Finally, a new branch of exponential time algorithms has been recently developed. Their interest is twofold. First, they aim at finding exponential time algorithms with small basis of the exponent, which can make them of practical use for instances of moderate size if the basis is close enough to 1. Then, the complexity analysis may serve to evaluate on how adequate are the tools used to describe the combinatorial structure of the problem, which stimulates further research. It is worth mentioning here that these approaches to NP-hard problems are in no way exclusive. They rather tend to complement and reinforce each other: observations on exact solutions for particular cases are incorporated into heuristics, heuristics are refined into approximation algorithms, approximations prove to be exact for particular instances, etc.

One of the methods used for tackling NP-hard problems on restrained graph classes is through graph decompositions. The basic idea is to decompose the vertex set or the edge set of a graph into clusters which satisfy certain conditions. The properties of this decomposition are such that

if one can solve the problem on the clusters, then these partial solutions can be joined to yield a global solution. Typically, the clusters are decomposed recursively, which gives the structure of a decomposition tree. If the structure of each cluster is simple enough, with respect to the sub-clusters it is composed of, then the problem can be efficiently solved.

The first kind of decomposition that attracted much attention was the modular decomposition, devised by Gallai in the sixties of the last century [55]. The cluster in modular decomposition is called a module. It is a set of vertices that are indistinguishable from outside the cluster, that is, all vertices of a module have exactly the same adjacent vertices outside the module. The modular decomposition of a graph is unique. Moreover, one can compute the modular decomposition in linear time (see, e.g. [35]). Given the modular decomposition of a graph, the difficulty of solving a problem resides in solving it for the (prime) subgraphs which cannot be further decomposed, that is, the only modules they strictly contain consist of single vertices. There are many results on solving NP-hard problems for graphs having good modular decompositions. Here, "good" means that the prime subgraphs are simple enough that the problem can be efficiently solved on them.

Tree decompositions were introduced in the early eighties of the last century by Robertson and Seymour [105, 104]. They defined tree decompositions and the associated quality measure of therewith in their work on graph minors. A closely related notion of a partial k-tree (G is a partial k-tree iff the treewidth of G equals k) was introduced ten years earlier by Dirac and Rose [106]. A good intuition on tree decompositions comes from chordal (also known as triangulated) graphs, ie. the graphs without chordless cycles of length greater than 3. The maximal cliques of a chordal graph can G be arranged as the nodes of a tree T, in a particular way. That is, for each vertex v of G, the nodes of T that contain v induce a subtree of T. T is called a clique tree of G. In a tree decomposition, the condition on nodes of T to be cliques of G is removed. The conjunction of the following three conditions gives the standard definition of a tree decomposition:

1. For every vertex of G, there is a node of T that contains it.

2. For every edge of G, there is a node of T that contains both its endpoints.

3. For every vertex of G, the set of nodes of T that contain it induces a subtree of T.

An edge of T, labeled with the intersection of the incident nodes, corresponds to a minimal separator of G. The width of a decomposition is the maximum cardinality of a node in the decomposition tree. Getting back to chordal graphs, an arbitrary chordal graph has a decomposition tree in which every bag is a clique; in particular, a tree has a decomposition tree in which every bag is an edge. In this way, any chordal graph G can be seen as a thickened tree, with nodes replaced by cliques; the clique number of G minus one gives the treewidth of G. In general, an arbitrary graph is of treewidth at most k if and only if it is a subgraph of a chordal graph of clique number at most $k+1$. This is why the treewidth is sometimes regarded as a measure of similarity to a tree. Tree decompositions proved to be very useful not only from purely combinatorial but also from algorithmic point of view. Most classical problems that are NP-hard in general case can be solved in polynomial time for graphs of constant bounded treewidth. Among these are graph coloring, traveling salesman, minimum dominating set, etc.

Modular decompositions and tree decompositions are complementary tools for handling algorithmic problems, since there are graph classes that have very good decompositions of one type but not of the other, in both ways. For example, the class of trees trivially admits tree decompositions of width 1, which is the best possible; whereas, any path of length $n-1$ for $n > 3$ is a prime subgraph,

so admits a very bad modular decomposition. Conversely, the class of complete graphs admits very good modular decompositions, without prime nodes at all; meanwhile, tree decompositions are very bad here, with the trivial decomposition of just one bag containing the whole vertex set meeting the optimum. Here come the clique decompositions [32, 34] (or, almost equivalently, NLC decompositions [114]) which, in a way, gather the good points of both methods mentioned above. That is to say that graphs that admit a good tree decomposition or a good modular decomposition also have a good clique decomposition. Even though the clique decompositions are a very recent invention, there have been a growing interest in the topic, with results generalizing the complexity results on graphs of bounded treewidth to the larger class of graphs of bounded cliquewidth. But there are two reasons that make tree decompositions algorithmically more interesting at the moment. The first is that there are no efficient algorithms computing clique decompositions. The other is that, even given a good clique decomposition, the algorithms using it are much slower than the similar ones using good tree decompositions.

Most classical NP-hard problems are solvable in polynomial time for graphs of bounded treewidth. In particular, there are results identifying the problems which are NP-hard in general but can be solved in linear time for graphs of bounded treewidth. The work of Courcelle [31], Arnborg et al.[4], Courcelle et al. [33] showed that all problems expressible in the extended monadic second order logic have this property. Although very interesting from the theoretical point of view, results based on formulations in terms of logic are of little use in applications, since the constants appearing in time complexity analysis of these algorithms are prohibitive. Therefore, from the practical point of view, solutions to real-life problems are rather based on ad-hoc dynamic programming [14, 68]. The dynamic programming on graphs of bounded treewidth proves useful also on many problems that do not fit in the logic framework mentioned above. The time complexities involved are usually of kind $\mathcal{O}(2^k n)$, where n is the number of vertices of the instance graph G and k is the width of the tree decomposition of G we have. This gives an idea of the upper bound on the decomposition width up to which this approach is reasonable.

Tree decompositions from the very beginning were a powerful tool for showing that certain problems are polynomial time solvable for particular instances. They also proved their value in the context of fixed parameter tractability and polynomial approximation schemes. For a couple of years, we have experienced a real flourishing of applications of tree decompositions in many areas of practical use. These are probabilistic networks, which often have small treewidth, constraint programming [62, 77, 78], many optimization problems [1, 75], etc. The weak point of tree decompositions comes from the fact that computing one of minimum width is NP-hard [3], even for restrained graph classes like the graphs of bounded degree [21], bipartite graphs, and co-bipartite graph [66]. Due to Bodlaender [15], there is an algorithm that, given a graph G and a constant k, decides in $\mathcal{O}(n)$ time if the treewidth of G is at most k. But the constant is super-exponential in k, which makes it useless in real-life applications even for very small values of k. What works well are algorithms computing the treewidth and a corresponding tree decomposition for restricted graph classes, like permutation graphs [19], circular-arc graphs [86, 112], bipartite chordal graphs [84], distance-hereditary graphs [27], HHD-free graphs [26], circle graphs [83, 86], weakly triangulated graphs [25]. For all these classes, the algorithms are based on the notions of chordal completion and minimal separator. Given an arbitrary graph G, a chordal completion (also known as a triangulation) H of G is a chordal super-graph of G on the same vertex set. H is a minimal chordal completion of G if no strict subgraph of H is a chordal completion of G. The treewidth of G can be defined as the minimum clique number minus one over all chordal completions of G. Clearly, a

chordal completion which realizes this optimum can always be found among the minimal chordal completions of G. The studies on the structure of minimal chordal completion have given tools used not only in the polynomial time exact computations of treewidth on particular graph classes mentioned above, but also in general approximation and heuristics.

As we mentioned above, tree decompositions have many advantages over clique decompositions, even though the latter are more general. Similar reasons make path decompositions, a particular case of tree decompositions, interesting. They are characterized by the condition on the decomposition tree to be a path. This simplification in structure allows more simple, elegant, time and space efficient algorithms than the ones based on general tree decompositions. Apart from this general algorithmic interest, there are other graph parameters directly related to the pathwidth that have been investigated and give motivation to further studies on the pathwidth. Among these are the vertex separation number, the interval thickness, the node search number, etc. See [16] for a survey. In particular, the node search number is one of the parameters analyzed in the context of graph searching games. Here the motivation comes from various network management tasks that can be modeled as a pursuit of a hostile mobile agent.

Like between the treewidth and the class of triangulated graphs, there is a similar relation between the pathwidth and the class of interval graphs. We mentioned above that an arbitrary chordal graph has a decomposition tree in which every bag is a clique. In fact, this characterizes the class of chordal graphs. In a similar way, a graph G is interval if and only if it has a decomposition path in which every bag is a clique. To continue with the analogies, given an arbitrary graph G, an interval completion H of G is an interval super-graph of G on the same vertex set. H is a minimal interval completion of G if no strict subgraph of H is an interval completion of G. Finally, the pathwidth of an arbitrary graph G can be computed as the minimum clique number minus one over all interval completions of G. Clearly, an interval completion which realizes this optimum can always be found among the minimal interval completions of G.

Interval graphs, on their own, have a long list of applications in areas like biology, chemistry, and archeology, and many NP-complete graph problems are solvable in polynomial time on interval graphs [61]. Specifically, the problem of adding edges to a given input graph to obtain an interval graph, called an *interval completion* of the input graph, arises in Physical Mapping of DNA [60, 96], Orthogonal Packing [42], and Sparse Matrix Computations [58]. For several applications, it is desirable to embed a given graph into an interval graph by adding as few edges as possible. Such an embedding is called a *minimum interval completion*, and the number of edges it contains is called the *profile* of the input graph. Both profile and pathwidth are well known and well studied graph parameters, and both are NP-hard to compute [56, 63]. There has been extensive work on computing these parameters for restricted graph classes [36], but our insight on how to handle arbitrary graphs is limited.

As a comparison, minimal chordal completions were studied and a polynomial time algorithm for computing them was given already in 1976 [107], even before it was proved that minimum fill (minimum chordal completion) is NP-hard to compute [116]. Since then many results have been added about minimal chordal completions [69, 74, 87], which are central in understanding and trying to solve minimum fill and treewidth problems. Minimal chordal completions have several quite different characterizations, and some of these have proved useful in trying to compute minimum fill and treewidth [25, 80], either by approximation algorithms [95] or by exact (fast) exponential time algorithms [49]. Following the history of chordal completions, our hope is that understanding and characterizing minimal interval completions will eventually lead to improved

exact or approximation algorithms for computing profile and pathwidth.

In this report we present the results of two years' work on minimal interval and proper interval completions in the context of pathwidth and bandwidth parameters. As to our knowledge, this reflects the current state of the art of the topic. There are three polynomial time algorithms computing minimal interval completions, based on substantially different aspects of the combinatorial structure of the problem [73, 110, 72]. There is one polynomial time algorithm computing minimal proper interval completions [101]. Finally, there is a polynomial time algorithm computing the pathwidth of circular-arc graphs [111].

Let us present the structure of this report.

Chapter 2 : Basis provides an introduction to the topic of minimal interval completion. It makes a review of basic notions of graph theory that we use. Then, it introduces the intersection graphs. In particular, it surveys properties of chordal, interval, proper interval and circular-arc graphs that prove useful in the study on minimal completions. The presentation starts with a unified characterization of these classes. Chordal, interval and circular-arc graphs are described as intersection graphs of connected subgraphs of a tree, a path or a cycle, respectively. It is followed with a description of two approaches to characterizing the above mentioned graph classes that are later used to define minimal completions. The first approach proceeds by the means of *connected decompositions*, a generalization of tree decompositions. The second uses *layouts*, permutations of the vertex set having particular properties. Finally, some other characterizations and useful lemmas are given. The chapter is closed with a short discussion on how to use connected decompositions or layouts to obtain the corresponding completions.

Chapter 3 : Minimal Proper Interval Completion Through Exploration presents the results of [101]. They consist in showing that a minimal proper interval completion can be obtained by choosing a particular permutation of the vertex set, called a *nice layout* (for proper interval completion) σ, and adding edges in order to make σ a *bicompatible ordering* of the resulting graph. First, the procedure of computing a proper interval completion based on a layout is given, together with the definition of a nice layout. Then, the notions of a moplex and of a moplexian vertex are introduced. They proved useful as definitions of "extremities" of a graph. Finally, a characterization of a family of nice layouts is given together with a $\mathcal{O}(n + m)$ time algorithm computing a minimal proper interval completion. The algorithm is a BFS exploration with a particular tie-break rule, that can be computed locally.

Even though [101] was not chronologically the first of our papers on interval completions, we decided to start the report with its description. The reason is that it provides a gentle introduction to some of the techniques that we use. The following chapters will develop these, and gradually add some more. As far as it was possible, we tried to keep a steady rhythm of introducing new notions.

Chapter 4 : Minimal Interval Completion Through Exploration presents the results of [110]. They consist in showing that a minimal interval completion can be obtained by choosing a particular permutation of the vertex set, called a *nice layout* (for interval completion) σ, and adding edges in order to make σ an *interval ordering* of the resulting graph. First, the procedure of computing an interval completion based on a layout is given, together with the definition of a nice layout. Then, a characterization of a family of nice layouts is given. For this purpose, some further properties of moplexes and moplexian vertices are given. In particular, the last vertex explored in a LexBFS exploration of the graph is described as a particular type of a moplexian vertex, with stronger "extremity" properties. Finally, the characterization is exploited in a $\mathcal{O}(nm)$

time algorithm computing a minimal interval completion. The algorithm is a BFS exploration with an additional tie-break rule, based on an additional LexBFS launched at each step.

Chapter 5 : Pre-order - clique path encoding describes the main tool conceived to tackle the problem of minimal interval completion in [73]. There, it is used as a compact encoding of all possible clique paths of an interval graph G. The pre-order permits to compute finer and finer path decompositions down to fixing a clique path of G. The elementary step consists in choosing a minimal separator S of G and computing the set of connected components $\mathcal{C}(S)$ of $G - S$. Each component $C_i \in \mathcal{C}(S)$ together with its neighborhood $N_G(C_i)$ yields a block $O_i = C_i \cup N_G(C_i)$. The blocks arranged on a path form a path decomposition of G. The pre-order relation tells if one block can be put between another and a block containing S in a path decomposition of G. It is a partial pre-order of width 2 (maximum cardinality of an anti-chain). A decomposition of the pre-order into two chains gives a path decomposition of G. These correspondences are explored first. Then, a more detailed analysis of the structure of the pre-order is given. This study is necessary for a good control over the pre-order, and is used in the main proof of [73]. The final part of the chapter is devoted to efficient learning of the pre-order. It can be achieved in linear time.

Chapter 6 : Minimal Interval Completion Through Decomposition presents the results of [73]. The chapter starts by showing that a minimal interval completion can be computed online. With vertices of G coming in an arbitrary order $(v_1, \ldots, v_n)$, at each step a minimal interval completion H_i of the graph $G_i = G[\{v_1, \ldots, v_i\}]$ is computed. The completion H_{i-1} computed at the previous step is not modified, only edges incident to v_i are added. This reduces the problem to computing a minimal interval completion of a graph G', for which the graph G obtained by removing a fixed vertex x is interval. First, a general description of the reduced problem is given. If the neighborhood of x is a clique in G', then there is a minimal separator S of G, such that making every vertex in S adjacent to x yields a minimal interval completion. The rest of the chapter covers the more difficult case where the neighborhood of x is not a clique. Then, the analysis of clique paths of G comes into play. It is shown that a minimal interval completion can be obtained by making x adjacent to every vertex that appears in a well defined interval of cliques in a particular clique path of G. A clique path which can be used for this purpose is called *nice*. Then the pre-order of Chapter 5 is used to refine path decompositions of G in order to find a nice clique path. Finally, a $\mathcal{O}(n^3 \log n)$ time algorithm computing a minimal interval completion is given.

Chapter 7 : Pathwidth of Circular-Arc Graphs presents the results of [111]. The chapter starts with an introduction to *folding*, a tool conceived to describe interval completions in terms of cliques of the original graph G. An algorithm is given, that computes an interval completion H of G based on a sequence $\mathcal{Q}$ of cliques of G which cover every edge of G (edge clique cover). The algorithm also produces a clique path of H based on $\mathcal{Q}$. This is called folding G by $\mathcal{Q}$. Then, some characteristics of circular-arc graphs are recalled. In particular, circular-arc graphs are described in terms of clique cycle intersection models, like interval graphs were described with clique paths before. The set of cliques in a clique cycle is an edge clique cover of G. This property is used to define folding a circular-arc graph as folding a clique cycle model. The main result of this chapter states that an interval completion of a circular-arc graph of minimum pathwidth can be obtained by a folding of a very simple structure. This combinatorial result is exploited in a $\mathcal{O}(n^2)$ time algorithm computing the pathwidth of circular-arc graphs.

Chapter 8 : Extraction of Minimal Interval Completion presents the results of [72]. Here, the tool of folding introduced in Chapter 7 is studied in detail in the case of folding interval graphs. The cliques in any clique path of an interval graph H_0 form an edge clique cover. So folding

an interval graph may be defined as folding a clique path. Then, the notion of a quasi-minimal interval completion H_2 of G is introduced. It is an interval completion which is not minimal, but there is no interval completion of G obtained by adding just one edge less than in H_2. Such completions are especially difficult to track down. To overcome this difficulty, the notion of folding is applied. A quasi-minimal interval completion H_2 of G is proved to be a folding of a minimal interval completion H_0 of G. The reverse problem of finding an *unfolding* of a quasi-minimal interval completion, that is to say, computing an interval completion H_1 of G strictly contained in H_2, is then studied. Finally, based on a more strict structure of a *reduced folding*, a polynomial time algorithm computing an unfolding is given.

The report is closed with **Conclusions and Perspectives**.

Chapter 2

Basis

Contents

2.1 Basic notions

A simple graph G is a couple (V, E), where V is an arbitrary set, called the *vertex set*, and E is an arbitrary set of two-element subsets of V, called the *edge set*. An edge $e = \{x, y\} \in E$, where x, y are vertices in V, for the sake of simplicity, will be sometimes denoted without the accolades by xy. A *non-edge* is a two-element subsets of V which is not in E. The graph $G' = (V, E')$, where E' is the set of non-edges of G, is the *complement* of G. For an edge xy, we say that x and y are *adjacent*, and that xy is *incident* to both x and y. xy is an *edge between* x and y, and between any sets X, Y such that $x \in X, y \in Y$. We also say that x and y are *neighbors*. The *neighborhood of a vertex* $x \in V$ in G, denoted by $N_G(x)$ is the set of its neighbors $\{y \mid xy \in E\}$. The *closed neighborhood* $N_G[x] = N_G(x) \cup \{x\}$. The *neighborhood of a set of vertices* $A \subseteq V$, denoted by $N_G(A)$, is the set of vertices in $V \setminus A$ that have a neighbor in A, i.e. $N_G(A) = \bigcup\{N_G(x) \mid x \in A\} \setminus A$. The *closed neighborhood* $N_G[A] = N_G(A) \cup A$. Vertices which are not adjacent are *independent*. A vertex v, for which $N_G[x] = V(G)$ is *universal*. We use the notation $V(G)$, $E(G)$ to refer to the vertex set and the edge set, respectively, of a given graph G, and $n = |V(G)|$, $m = |E(G)|$ for their cardinalities. If there is no ambiguity as to which graph we refer, its name may be omitted in the notation. If every two vertices in G are neighbors then G is a *complete graph*. The complement of a complete graph is an *empty graph*. If the vertex set of G can be partitioned into two subsets A, B in a way that no edge of G if a subset of either A or B, then G is *bipartite*. If the neighborhood of every

vertex in A equals B, and vice-versa, then G is *complete bipartite*graph!complete bipartite on the sets A, B. The sets A, B are in this case called the *color classes* of $G = (A, B, E)$. A complete bipartite graph on the sets A, B, with $|A| = a$ and $|B| = b$, is denoted by $K_{a,b}$. A complete graph on n vertices is denoted by K_n.

Given another graph $G' = (V', E')$, where $V' \subseteq V$ and $E' \subseteq E$, G' is a *subgraph* of G. If E' is the set of edges incident only to vertices in V' then G' is the subgraph of G *induced* by V'; this fact is denoted by $G' = G[V']$. If $V' = V$ and $E' \subseteq E$, then G' is a *spanning subgraph* of G, denoted by $G' \subseteq G$. A sequence of vertices $W = (x_1, \ldots, x_k)$, in which every two consecutive vertices are neighbors in G is a *walk*. Equivalently, W can be treated as a graph (a subgraph of G) with the vertices in W as the vertex set and an edge for every par of vertices that appear consecutively in W. If all vertices of a walk are different then it is a *path*. If the only equality is between the first and the last vertex of a walk then it is a *cycle*. An edge between two vertices which do not appear consecutively in W is a *chord*. A walk without a chord is *chordless*. We say that W *joins* x_1 with x_k. The *length* of a walk $(x_1, \ldots, x_k)$ is $k - 1$. We say that a graph $G = (V, E)$ is *connected* if for all $x, y \in V$ there exists a path joining them. A set of vertices which induces a connected subgraph of G, and is inclusion maximal for this property is a *(connected) component* of G. A set of vertices which induces a complete graph is a *clique*. A set inclusion maximal for this property is a *maximal clique*. A set of vertices which induces an empty graph is a *stable set*. Three independent vertices form an *asteroidal triple* (abbreviated to AT) if any two of them are joined by a path that does not intersect the closed neighborhood of the third.

Given a set of vertices $S \subseteq V$, the graph $G - S$ is the subgraph of G induced by $V \setminus S$, i.e. the graph $G' = (V', E')$ where $V' = V \setminus S$ and E' is the set of edges not incident to any vertex in S.

2.1.1 Minimal separators

If there are two vertices $x, y \in V \setminus S$ that are joined by a path in G and not joined by any path in G' then S is a *separator*. In this case we say that S *separates* x from y. Moreover, if S is inclusion minimal for this property then it is a *minimal x, y-separator*. In general, S is a *minimal separator* if there are some x, y such that S is a minimal x, y-separator. The *components associated* to S are the connected components of $G - S$. This set is denoted by $\mathcal{C}(S)$. $C \in \mathcal{C}(S)$ is a *full component* associated to S if $N(C) = S$.

It is a well known fact that, for any minimal separator, there exist two associated full components. A *block* $O \subseteq V$ is the union of a minimal separator S with a full component C associated to S. S is called the *separator bordering* O. Notice that, given a block O, the bordering separator $S = N_G(V \setminus O)$.

Given a minimal separator S, a component $M \in \mathcal{C}(S)$ which is a clique and the neighborhood of every vertex in M contains S is a *moplex*. An element of a moplex is a *moplexian vertex*.

2.1.2 Ordered partitions

Given two tuples $\mathcal{O}' = (O_1, \ldots, O_k)$, $\mathcal{O}'' = (O_{k+1}, \ldots, O_{k+l})$ we write $\mathcal{O}' \bullet \mathcal{O}''$ to denote their concatenation $\mathcal{O} = (O_1, \ldots, O_k, O_{k+1}, \ldots, O_{k+l})$. $\overleftarrow{\mathcal{O}}$ denotes the tuple obtained from $\mathcal{O}$ by reversing the order of elements. A tuple of disjoint subsets of V, $\mathcal{O} = (O_1, \ldots, O_k)$ whose union is exactly V is called an *ordered partition* of V. A *refinement* of $\mathcal{O}$ is an ordered partition $\mathcal{O}'$ obtained by replacing each set O_i by an ordered partition $(O_i^1, \ldots, O_i^l)$ of O_i, that is to say, replacing $(O_1, \ldots, O_i, \ldots, O_k)$ with $(O_1, \ldots, O_i^1, \ldots, O_i^l, \ldots, O_k)$. We write $\mathcal{O}' \preccurlyeq \mathcal{O}$. Given an ordered partition $\mathcal{O} = (O_1, \ldots, O_k)$,

any tuple $\mathcal{O}' = (O_1, \ldots, O_j)$, with $0 \leq j \leq k$, is called a *prefix* of $\mathcal{O}$. In the particular case where $\mathcal{O} = (O_1)$, we simply write O_1. Moreover if O_1 is formed by a single vertex x, we write x instead of $\{x\}$. A permutation of V is a particular case of an ordered partition.

For any structure $\mathcal{X}$ on a subset of the vertex set, we use $\mathrm{V}(\mathcal{X})$ to denote the set of vertices in the structure. In particular for an ordered partition $\mathcal{O} = (O_1, \ldots, O_k)$, $\mathrm{V}(\mathcal{O})$ denotes $\bigcup\{O_i \mid 1 \leq i \leq k\}$. Given a tuple $\sigma = (t_1, \ldots, t_k)$, we consider it as a linear order and sometimes write $t_i \leq t_j$ or say that t_i is smaller than t_j (with respect to σ), for any $i \leq j$. Notice that it also applies to a path, taken as a sequence of vertices. Moreover, given t_a, t_b we speak of the interval $[t_a : t_b]$ as of the set $\{t_i \mid a \leq i \leq b\}$. Throughout this report we consider finite, simple graphs. Moreover, we assume they are connected, since in the non-connected case each component can be treated separately.

2.2 Intersection graphs

In this section we introduce four classes of intersection graphs that are of particular interest of us, namely, chordal, interval, proper interval and circular-arc graphs. These classes share a number of interesting properties that let us see them in a unified way. Two of their aspects we describe here in detail, because they proved very useful in analyzing minimal completions. The first aspect is that all these classes can be characterized through connected decompositions, which is a straightforward consequence of the definitions given in the next subsection. The other is that they can be characterized by vertex layouts. In the last subsection, we give some other characterizations that prove useful throughout the work on these classes.

2.2.1 Intersection model

The *intersection graph* of a family V of n sets is the graph $G = (V, E)$, where the vertices are the sets and the edges are the pairs of sets that intersect. Every graph is the intersection graph of some family of sets. In fact, general topology is not needed, since a stronger result holds:

Theorem 2.2.1 ([92]). *Every graph $G = (V, E)$ is the intersection graph of a family $\mathcal{W} = \{W_v \mid v \in V\}$ of subsets of the vertex set of a graph $H = (W, F)$ which induce connected subgraphs of H. The graph H is called the* host graph *and $(H, \mathcal{W})$ is called the* intersection model *of G. For any $v \in V$, the set W_v is called the* representation *of v in H.*

In this work, we focus on this graph theoretic framework. We study the following graph classes.

Definition 2.2.2 (chordal graph). *G is an* chordal graph *if it has an intersection model $(H, \mathcal{W})$, where H is a tree.*

Since for each vertex $v \in V(G)$ the corresponding $W_v \in \mathcal{W}$ induces a connected subgraph of H, which is a tree, G is represented by intersecting subtrees of H.

Definition 2.2.3 (interval graph). *G is an* interval graph *if it has an intersection model $(H, \mathcal{W})$, where H is a path.*

Clearly, the elements of $\mathcal{W}$ induce subpaths of H, and are called the intervals of the model. The first and the last element of an interval W_i are called the *endpoints* of W_i.

Definition 2.2.4 (proper interval graph). *G is a* proper interval graph *if it has an intersection model $(H, \mathcal{W})$, where H is a path and no interval is properly contained in another. That is to say, if one interval is contained in another, then they share an endpoint.*

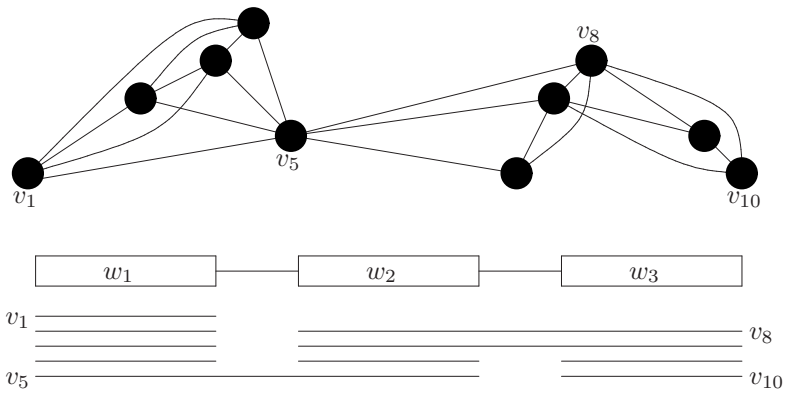


Figure 2.1: A proper interval graph and an interval model.

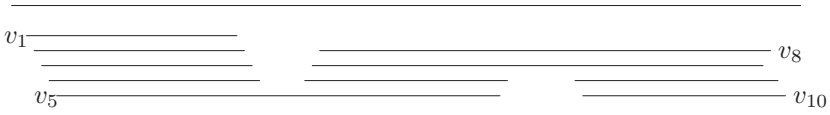


Figure 2.2: A slight modification of intervals to avoid strict inclusion.

Definition 2.2.5 (circular-arc graph)**.** *G is a* circular-arc graph *if it has an intersection model $(H, \mathcal{W})$, where H is a cycle.*

So a circular-arc graph is represented by a family of arcs on a cycle, that is a family of paths and, possibly, the whole cycle.

The original definitions of these classes were quite different. Chordal graphs, also called triangulated graphs, were first defined as the class of graphs containing no chordless cycles of length greater than three (see [61]). It was some time later that Gavril [57] proved they can be characterized as intersection graphs of subtrees of a tree. Interval and proper interval graphs were first defined as intersection graphs of intervals on a line, without proper inclusion in case of proper interval graphs. Also the circular-arc graphs were first defined with geometrical objects: arcs on a cycle. In this document we use the definitions given above in purpose of unifying the approaches and giving a common ground for our analysis. We emphasize that our definition of proper interval graphs is equivalent to the classical one. The definition using the proper containment of intervals as the set inclusion with disjoint sets of endpoints is equivalent, since a slight modification of the intervals yields a model that satisfies the traditional definition. Indeed, if one interval is strictly included in another, but they share an endpoint, w.l.o.g. assume it is the starting point, then it is enough to slightly enlarge the shorter interval to the left (by some ϵ small enough not to add new intersections) to make the intervals incomparable by inclusion. This gives the following lemma.

Lemma 2.2.6. *Proper interval graphs are exactly the intersection graphs of intervals on a line which have an intersection model in which no interval is strictly contained in another.*

Notice that all these families are hereditary, that is to say, they are closed under taking induced subgraphs. Indeed, let $(H, \mathcal{W})$ be an intersection model of a graph G in one of the above mentioned

classes. Given a subset $V' \subseteq V$, one may restrict $\mathcal{W}$ to $\mathcal{W}' = \{W_v \in \mathcal{W} \mid v \in V'\}$. This yields the pair $(H, \mathcal{W}')$, which is an intersection model of $G[V']$. $G[V']$ is in the same class as G.

There are some inclusions among these classes. Proper interval graphs are interval graphs. Interval graphs are both chordal and circular-arc graphs.

2.2.2 Connected decomposition

Let $G = (V, E)$ be a graph and let $(H, \mathcal{W})$ be an intersection model of G, where $H = (W, F)$. For any vertex $w \in W$ of H, any two vertices v_1, v_2 of G whose representation in H contains w are adjacent in G. The corresponding set $V_w = \{v \in V \mid w \in W_v\}$ is a clique in G. Moreover, every edge of G is a subset of such a clique. So the family $\mathcal{V} = \{V_w \mid w \in W\}$ is an edge clique cover of G.

Definition 2.2.7 (edge clique cover)**.** *Let $\mathcal{X} = \{X_1, ..., X_k\}$ be a set of subsets of V such that X_i, $1 \leq i \leq k$, is a clique in $G = (V, E)$. If, for every $\{v_i, v_j\} \in E$, there is some X_p such that $\{v_i, v_j\} \subseteq X_p$, then $\mathcal{X}$ is called an* edge clique cover *of G.*

It is often convenient to identify the vertices of a host graph H with the corresponding members of the family $\mathcal{V}$. In this way, for any G which is chordal, interval, proper interval or circular-arc graph, the corresponding host graph H is a clique connected decomposition of G.

Definition 2.2.8. *A* connected decomposition *of an arbitrary graph $G = (V, E)$ is a graph $D = (\mathcal{X}, A)$, where $\mathcal{X}$ is a family of subsets of V called* bags *and A is any set of edges on $\mathcal{X}$, such that the following three conditions are satisfied.*

1. *Each vertex $v \in V$ appears in some bag.*
2. *For every edge $\{v_i, v_j\} \in E$ there is a bag containing both v_i and v_j.*
3. *For every vertex $v \in V$, the bags containing v induce a connected subgraph of D.*

$D = (\mathcal{X}, A)$, a connected decomposition of G, is a clique connected decomposition *of G if $\mathcal{X}$ is an edge clique cover of G.*

In case of a chordal graph G, the number of maximal cliques is at most equal the number of vertices, so it is plausible to analyze the edge clique cover given by the set of maximal cliques. In general, it is not the case for a circular-arc graph, since it may have exponentially many maximal cliques. Many authors restrict the name of clique tree or clique path to maximal clique connected decompositions. Nevertheless, the maximality condition on bags is not necessary to have most of interesting properties. So we use the relaxed versions given below.

Definition 2.2.9. *A connected decomposition with D being a tree, path or a cycle is a* tree, path or cycle decomposition*, respectively. A clique connected decomposition with D being a tree, path or a cycle is a* clique tree, path or cycle*, respectively.*

Lemma 2.2.10 ([61])**.** *G is a chordal, interval, circular-arc graph iff it has a clique tree, path or cycle, respectively. G is a proper interval graph iff it has a clique path in which there is no proper containment of intervals induced by vertices of G (see Definition 2.2.4).*

2.2.3 Layouts

A *layout* of G is a permutation $\sigma = (v_1, \ldots, v_n)$ of its vertex set (see Subsection 2.1.2). A vertex x whose neighborhood is a clique is a *simplicial vertex*. A layout $\sigma = (v_1, \ldots, v_n)$ in which v_i is simplicial in $G[V_i]$, for every $1 \leq i \leq n$, $V_i = \{v_i, \ldots, v_n\}$, is called a *perfect elimination ordering*. A perfect elimination ordering σ for which also $\overleftarrow{\sigma}$ is a perfect elimination ordering is a *bicompatible ordering*. These notions are related to characterizations of the above mentioned graph classes that proved very useful in the work on minimal completions.

It is well known that a graph is chordal if and only if it has a perfect elimination ordering.

Theorem 2.2.11 (perfect elimination ordering, [54]). *$G = (V, E)$ is a chordal graph iff it has a layout $\sigma = (v_1, \ldots, v_n)$ such that:*

$$\forall i < j < k : v_i v_j \in E \wedge v_i v_k \in E \Rightarrow v_j v_k \in E.$$

Such a layout is a perfect elimination ordering.

A graph is interval if and only if it has an interval ordering.

Theorem 2.2.12 (interval ordering, [97]). *$G = (V, E)$ is an interval graph iff it has a layout $\sigma = (v_1, \ldots, v_n)$ such that:*

$$\forall i < j < k : v_i v_k \in E \Rightarrow v_i v_j \in E.$$

Such a layout is an interval ordering.

Notice that for an interval ordering σ, the reverse ordering $\overleftarrow{\sigma}$ is a perfect elimination ordering.

Remark 2.2.13. *Let $\sigma = (v_1, v_2 \ldots, v_n)$ be an interval ordering of an interval graph $G = (V, E)$. An interval model of G on the path $(1, \ldots, n)$ can be obtained by associating to each vertex v_i the interval $[i : j]$, where $j \geq i$ is the largest index such that $\{v_i, v_j\} \in E$, or the interval $[i : i]$ if no such j exists.*

Conversely, given an interval model of the graph G, we obtain an interval ordering by ordering the vertices according to the left end-point of their intervals, from left to right. Ties can be broken arbitrarily. For technical reasons, in this work, we decide to use the order of the right-ends as a tie-break, from left to right, too.

Given an interval model, a clique path can be obtained by traversing the model from left to right and, at each point p where an interval finishes, adding the clique of intervals intersecting p if it is not included in the (maximal) clique added right before. If σ is an interval ordering of G, let $P(G, \sigma)$ denote the clique path obtained in that way.

A graph is proper interval if and only if it has a bicompatible ordering.

Theorem 2.2.14 (bicompatible ordering, [98]). *$G = (V, E)$ is an interval graph iff it has a layout $\sigma = (v_1, \ldots, v_n)$ such that:*

$$\forall i < j < k : v_i v_k \in E \Rightarrow v_i v_j \in E \wedge v_j v_k \in E.$$

Such a layout is a bicompatible ordering.

Notice that for a bicompatible ordering σ, both σ and revere ordering $\overleftarrow{\sigma}$ are interval orderings.

Finally, a graph is circular-arc if and only if it has a circular ordering.

Theorem 2.2.15 (circular ordering)**.** *$G = (V, E)$ is a circular-arc graph iff it has a layout $\sigma = (v_1, \ldots, v_n)$ such that:*

$$\forall i < k : v_i v_k \in E \Rightarrow (\forall j : i < j < k \Rightarrow v_i v_j \in E) \vee (\forall j : k < j \vee j < i \Rightarrow v_j v_k \in E).$$

Such a layout is a circular-arc ordering.

2.2.4 Other characterizations and properties

Theorem 2.2.16 ([61])**.** *G is a chordal graph iff it has a* maximal clique tree*, that is a clique tree on the set of maximal cliques of G.*

Theorem 2.2.17 ([61])**.** *G is an interval graph iff it has a* maximal clique path*, that is a clique path on the set of maximal cliques of G.*

Remark 2.2.18. *Let $G = (V, E)$ be an interval graph on n vertices, and let P_G be any maximal clique path of G. For the ease of description, we arbitrarily fix an orientation: in what follows, we assume that one of the endpoints of a clique path is chosen as the first (or the leftmost) bag, and denoted by K_1. So the maximal cliques of G can be enumerated $(K_1, \ldots, K_k)$ according to their order of appearance in P_G. K_k is called the last (or the rightmost) bag of P_G. Based on an orientation of P_G, G can be encoded in linear space by storing, for each vertex $v \in V$, the index of the first, denoted by $l(v)$, and the last, denoted by $r(v)$, maximal clique containing v. Indeed, it is linear, since the number of maximal cliques of G is bounded by n (see [61]). Such an encoding is called an* integer interval encoding *of G.*

Based on the characterization by maximal clique path, there are some very useful lemmas. The first of them is sometimes called the "intersection property of clique trees". It is very often used in discussions of clique trees.

Lemma 2.2.19 ([11])**.** *Given a clique tree T of a chordal graph G, for every pair K, K' of nodes in T, the intersection $K \cap K'$ is contained in every node on the path joining K to K' in T.*

The next lemma states that, given any clique tree T of a chordal graph G, every separator S of G is the intersection of two maximal cliques K, K' adjacent in T. In this way, the edge of T joining K and K' "represents" S.

Lemma 2.2.20 ([11])**.** *Let G be a chordal graph with a maximal clique tree T. Let $S \neq \emptyset$ be a set of vertices of G. S is a minimal separator of G iff $S = K \cap K'$ for some $KK' \in E(T)$. In particular, every minimal separator is a clique.*

The following two lemmas state that both a maximal clique K and a minimal separator S separate vertices of $G - K$ (resp. $G - S$) that appear in different subtrees obtained by removing the node corresponding to K (resp. an edge representing S) from a clique tree T of G.

Lemma 2.2.21 ([11])**.** *Let G be a chordal graph with a maximal clique tree T. Let Ω, Ω' be two maximal cliques of G adjacent in T. Consider the two subtrees of T obtained by removing the edge between the nodes Ω and Ω'. Let T_Ω be the subtree containing Ω and $T_{\Omega'}$ the subtree containing Ω'. We denote by V_Ω and $V_{\Omega'}$ the union of maximal cliques in T_Ω or $T_{\Omega'}$, respectively. Then the minimal separator $S = \Omega \cap \Omega'$ separates any vertex of $V_\Omega \setminus S$ from any vertex of $V_{\Omega'} \setminus S$.*

Lemma 2.2.22 ([11]). *Let G be a chordal graph with a maximal clique tree T. Let Ω be a maximal clique of G which is not a leaf in T. Consider $T_1, \ldots, T_k$, the subtrees of T obtained by removing the nodes Ω. We denote by V_i, $1 \leq i \leq k$, the union of maximal cliques in T_i. Then the maximal clique Ω separates any vertex of $V_i \setminus \Omega$ from any vertex of $V_j \setminus \Omega$, $1 \leq i < j \leq k$.*

The next lemma states that a minimal separator S of a chordal graph G partitions the set of maximal cliques of G into subsets corresponding to blocks associated to S.

Lemma 2.2.23. *Let K be a maximal clique of a chordal graph G. For any minimal separator S of G there is a component C associated to S such that $K \subseteq S \cup C$.*

Notice, that a maximal clique path is also a maximal clique tree. So, among others, the Lemmas 2.2.20 and 2.2.21 and 2.2.22 hold for an interval graph G with a clique path P.

Definition 2.2.24. *Three independent vertices form an* asteroidal triple *if any two of them are joined by a path that does not intersect the closed neighborhood of the third.*

We claimed that proper interval graphs are interval, and that interval graphs are chordal. The following theorems give the precise characterizations.

Theorem 2.2.25 ([22]). *G is an interval graph iff G is chordal and asteroidal-triple free.*

Theorem 2.2.26 ([61]). *G is a proper interval graph iff G is an interval graph without $K_{1,3}$ as an induced subgraph.*

2.3 Completion

Definition 2.3.1. *Given a graph $G = (V, E)$, a chordal, interval, proper interval or circular-arc super-graph $G' = (V, E')$ is called a* chordal, interval, proper-interval or circular-arc completion *of G, respectively. If no proper subgraph of G' has this property then G' is a* minimal completion. *The edges in $E' \setminus E$ are called* fill edges.

Definition 2.3.2. *The* treewidth, pathwidth, bandwidth or cyclewidth *of a graph is the minimum clique number minus one over all its chordal, interval, proper interval or circular-arc completions, respectively.*

Notice that, for each of the parameters mentioned above, optimal completions can be found among minimal completions.

Given a graph G and a tree decomposition T of G, the graph obtained from G by adding the edges necessary to make every bag a clique is a chordal graph. This operation of turning a set of vertices into a clique by adding the necessary edges is called *filling*. In a similar way, a path decomposition can be used to obtain an interval super-graph of G, and a cycle decomposition can be used to obtain a circular-arc super-graph of G. In general, given a graph G and a connected decomposition H, we denote the corresponding completion by $\texttt{Fill}(G, H)$.

A completion can also be obtained based on a layout. Given a graph G and a layout σ of G, the graph G' obtained by enforcing the characterization of perfect elimination ordering to hold for σ on G' is a chordal graph. This enforcing can be done by processing the vertices of G in the order given by σ and, at the moment of processing v_i, adding fill-edges necessary for the set of neighbors of v_i with bigger indices to be a clique. In a similar way, enforcing an interval ordering or a bicompatible

ordering defines a unique completion into an interval or proper interval graph, respectively. In this work we focus on completion into interval and proper interval graphs.

Much research has been devoted to the study of minimal chordal completions, also known as triangulations, especially with association to the treewidth parameter. For many years there were no results on the analogous problems of minimal interval and proper interval completion. One reason behind this may be that the minimality in case of interval and proper-interval graphs is more elusive. Namely, a triangulation H of G is minimal if and only if there is no edge $e \in E(H) \setminus E(G)$ such that $H - e$ is a chordal graph. This simple characterization was used in most results on minimal triangulations, whereas for interval and proper-interval graphs the analogues simply do not hold. It happens that H is an interval completion of G, such that, for any edge $e \in E(H) \setminus E(G)$, $H - e$ is not an interval graph, and still H is not a minimal interval completion. The same for proper interval completions.

This work presents the current knowledge on interval and proper-interval completions, based on connected decompositions and layouts, in the context of related graph parameters.

Chapter 3

Minimal Proper Interval Completion Through Exploration

Contents

In the previous chapter, section 2.3, we mentioned two approaches to computing an interval completion of an arbitrary graph G. The first takes a path decomposition P of G and turns all the bags into cliques, thus creates a clique path decomposition of an interval completion H of G. For a proper interval completion an additional condition on P is needed that there is no proper inclusion of intervals corresponding to vertices of G in P (see Definition 2.2.4 and Lemma 2.2.10). Unfortunately, we do not know how to exploit this approach in order to compute a minimal proper interval completion. Nevertheless, we mention it here because path decompositions were used in our first, successful, attack on the problem of minimal interval completion, presented in Chapter 6. After that, we tried to compute minimal proper interval completions with similar tools, but failed. And this failure made us search for other tools, and eventually helped discover the other approach to minimal completions, namely through layouts. In this chapter we present how to compute a layout of an arbitrary graph, such that adding fill edges in order to make it a bicompatible ordering yields a minimal proper interval completion. It is done with a linear time algorithm computing a minimal proper interval completion of an arbitrary graph. In Chapter 4 we will show how to extend this algorithm to yield minimal interval completions.

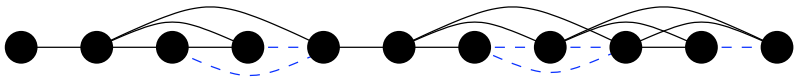


Figure 3.1: $\texttt{PIG}(G, \sigma)$ - the fill edges in dashed line-style.

3.1 ProperInterval layout

Let us put the discussion on using a layout to compute a minimal proper interval completion into formal definitions (see Subsection 2.2.3). First we define an operator that takes an arbitrary graph G and a permutation of the vertex set σ and yields a graph H by adding to G all the edges necessary for σ to be a bicompatible ordering of H.

Definition 3.1.1. *Let $G = (V, E)$ be an arbitrary graph and $\sigma = (v_1, \ldots, v_n)$ be a layout of G. The graph $\texttt{PIG}(G, \sigma) = (V, E')$ is defined by*

$$E' = \{v_j v_k \mid \exists i, l : i \leq j < k \leq l \wedge v_i v_l \in E\}.$$

We say that the graph $\texttt{PIG}(G, \sigma)$ is defined by σ.

Informally speaking, this operation corresponds to taking each edge e of G and turning into a clique the set of vertices "covered" by e, that is to say, the vertices which appear in σ between the endpoints of e (inclusive).

The following lemma, which simply states that the graph yielded by this operation is a proper interval graph, is a direct consequence of Theorem 2.2.14.

Lemma 3.1.2. *$\texttt{PIG}(G, \sigma)$ is a proper interval graph.*

What makes this operator particularly interesting is that any minimal proper interval completion of an arbitrary graph G can be obtained by applying PIG on some layout σ of G.

Theorem 3.1.3. *Let $G = (V, E)$ be an arbitrary graph and $G' = (V, E')$ be a minimal proper interval completion of G. Then there is a layout σ of G such that $G' = \texttt{PIG}(G, \sigma)$.*

Proof. By Theorem 2.2.14, there is a ordering σ of V bicompatible for G'. As a straight consequence of Definition 3.1.1 $E(\texttt{PIG}(G, \sigma)) \subseteq E(G')$. By Lemma 3.1.2, $\texttt{PIG}(G, \sigma)$ is also a proper interval graph. Thus, by minimality of G', we deduce that $E(\texttt{PIG}(G, \sigma)) = E(G')$. □

Therefore we can regard the computation of a minimal proper interval completion of G as a process mining the set of layouts of G in search for a nice one.

Definition 3.1.4. *An layout $\sigma = (v_1, \ldots, v_n)$ is called* nice *if $\texttt{PIG}(G, \sigma)$ is a minimal proper interval completion of G. Any prefix $(v_1, \ldots, v_k)$, $k \leq n$ of a nice layout is called a* nice prefix.

As it is usually the case with layouts, we compute them in an incremental way, starting with some well chosen vertex and then, at every step, computing the next vertex that should be appended to the prefix computed so far. This is why we give a name not only to a complete layout that yields a minimal proper interval completion through PIG, but also to every prefix that can be extended to such a layout. Maintaining nice prefixes we are sure that our algorithm never enters a dead-end - it is always possible to proceed without backtracking.

3.2 Moplex and LexBFS

Given an arbitrary graph G, we need to choose the first vertex for a layout σ such that $H = \texttt{PIG}(G, \sigma)$ will be a minimal proper interval completion of G. As mentioned before, see Remark 2.2.13, any interval ordering, in particular a bicompatible ordering, can be transformed into an interval model. In particular, the first vertex of the layout corresponds to the leftmost interval of a model obtained in this way. Therefore, we deal with some kind of extremity of the graphs in question. Berry and Bordat, in their research on graph extremities, defined the notion of a moplex which has proved to be very useful for our approach. As we will show in the next section, a moplexian vertex is a good choice for the first vertex of a nice layout. A prefix consisting of a single moplexian vertex is nice, i.e. it can be extended to a layout of G which yields, through the operator PIG, a minimal proper interval completion of G. Another crucial feature of a moplexian vertex is that it can be found in linear time.

Let us shortly introduce a few notions concerning moplexian vertices. The original work of Berry and Bordat used terminology that is out of the scope of this report. For this reason, we give a slightly different, but equivalent, definition.

Definition 3.2.1 ([7]). *Let S be a minimal separator (see Subsection 2.1.1) and let $M \in \mathcal{C}(S)$ be a component associated to S which is a clique. If the neighborhood of every vertex in M contains S, then M is a* moplex. *An element of a moplex is a* moplexian vertex.

They proved that a moplexian vertex always exists and can be found efficiently.

Theorem 3.2.2 ([7]). *Every graph has a moplexian vertex. Such a vertex can be found in $O(n+m)$ time. More precisely, the algorithm LexBFS (see Figure 3.2) ends on a moplexian vertex.*

In general, an exploration (search) algorithm visits all vertices of the graph and will visit a new vertex only if it is adjacent to some previously visited vertex. This does not indicate the rules on how to choose the next vertex among the neighborhood of the vertices explored so far. The two fundamental strategies are the *Breadth-First Search (BFS)* and the *Depth-First Search (DFS)*. The BFS strategy picks the next vertex among the neighbors of the first visited vertex that still has some neighbors to explore. The DFS strategy picks the next vertex among the neighbors of the last visited vertex that still has some neighbors to explore. But even with these rules, there may be some ties to be broken. The exploration algorithm that has earned the most attention over the last years is probably the *Lexicographic BFS (LexBFS)* algorithm. Here the strategy to pick the next vertex among the neighbors of the first visited vertex that still has some neighbors to explore is refined by adding additional filters coming from the second, third, and so on until the last vertex visited so far. This may be seen as a procedure numbering the vertices from n to 1 as they are visited and appending the number i, when the vertex v is numbered with i, to the label of each non-visited neighbor of v. Then the next vertex is the one having the lexicographically largest label among non-visited vertices (see Figure 3.2). Thus the name of the strategy. The order in which LexBFS visits the vertices is called a *LexBFS ordering*. The LexBFS algorithm was introduced by Rose, Leuker and Tarjan [107]. Their application was to recognize a chordal graph by finding a perfect elimination ordering, since the reverse of a LexBFS ordering is a perfect elimination ordering, given a chordal graph G. See [30] for a survey on LexBFS.

Algorithm `LexBFS`
Input: $G = (V, E)$ connected
Output: a numbering of the vertices from n to 1

// init
for each vertex x **do**
    x is marked "unnumbered"
    $lab(x) := \emptyset$
// main loop
for $i := n$ downto 1 **do**
    pick an unnumbered x with maximum label according to the lexicographic order
    give the number i to x
    for each unnumbered neighbor y of x **do**
        add the number i at the end of $lab(y)$

Figure 3.2: The `LexBFS` algorithm.

3.3 Nice layouts and nice prefixes

We already mentioned that any moplexian vertex constitutes a nice prefix. In this section we justify this claim and give a complete description of a family of nice layouts. The condition we give is sufficient but not necessary. There exist nice layouts which do not fit within our characterization.

We want to compute a layout σ of an arbitrary graph G such that the graph $H = \texttt{PIG}(G, \sigma)$ is a minimal proper interval completion. Our characterization tells, for any nice prefix ρ that starts with a moplexian vertex, in what set we should choose the next vertex to be added to the prefix in order to be sure that the new prefix is also nice. The main line of defending this solution is that any choice which does not follow the characterization implies the presence of a fill edge in the resulting completion which is not added by our algorithm. Then we prove that all choices that satisfy the characterization define minimal proper interval completions.

For a brief description of a layout σ satisfying our characterization we may say that it is a BFS ordering starting from a moplexian vertex, with a special tie-break rule. It is useful to recall here the standard color code for BFS algorithms, with the already explored vertices marked as black, the unexplored neighbors of explored vertices marked as gray, and the other vertices marked as white. Since we deal with a BFS ordering, given the current prefix ρ, the exploration continues with the neighbors of the first vertex in ρ, denoted $\text{First}(\rho)$, that still has some unexplored neighbors. The tie-break rule says that we should first choose a vertex that has an inclusion minimal set of white neighbors.

3.3.1 Choosing a first vertex

Let $G = (V, E)$ be an arbitrary graph. The following proposition shows that any moplexian vertex $v_1 \in V$ constitutes a nice prefix. In fact, for any proper interval completion H' of G that does not have a bicompatible ordering starting with v_1, the neighborhood of v_1 in H' is strictly bigger than in G. Whereas, there is a layout σ starting with v_1 such that $\texttt{PIG}(G, \sigma)$ preserves the neighborhood

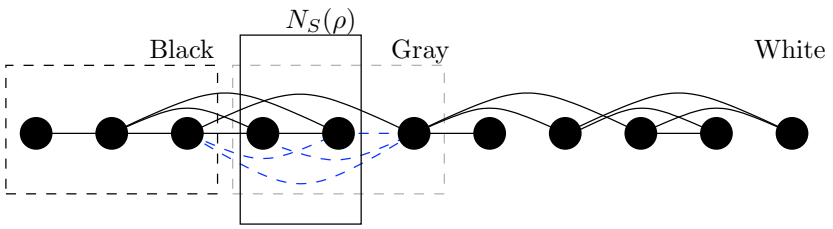


Figure 3.3: BFS color code and the strong neighborhood of ρ.

of v_1. Thus, putting v_1 first we preserve a non-edge which otherwise becomes a fill edge. In other words, any completion which does not have a bicompatible ordering starting with v_1 modifies the neighborhood of v_1.

Proposition 3.3.1. *Let M be a moplex of G and $v \in M$. There exists an nice layout σ starting with v such that the neighborhood of v in $\texttt{PIG}(G, \sigma)$ is exactly the neighborhood of v in G. Moreover, for any minimal interval completion H of G such that $N_G(v) = N_H(v)$, there exists a layout σ', starting with v and such that $H = \texttt{PIG}(G, \sigma')$.*

Proof. Let M be a moplex such that $v \in M$ (actually this moplex is unique [7]) and let H be the graph obtained from G by filling $V \setminus M$. We first show that H is a proper interval graph. Let $S = N(M)$. By definition of a moplex and by construction of H, the graph H is formed by two cliques, namely $M \cup S$ and $V \setminus M$. Their intersection is exactly S. Clearly, H is an interval graph. Moreover it has no independent set of size greater than 2, in particular it has no induced $K_{1,3}$. Hence H is interval and claw-free, so H is a proper interval graph by Theorem2.2.26. In particular there is a minimal proper interval completion of G contained in H.

Consider any minimal proper interval completion H' of G such that $N_G(v) = N_{H'}(v)$ (H' exists by the previous remark). By Theorem 2.2.14, there exists a layout σ' such that $H' = \texttt{PIG}(G, \sigma')$. If all vertices appearing before v in σ' are elements of M, we can permute v and the first element of σ' without changing the graph $\texttt{PIG}(G, \sigma')$. Similarly, if all vertices appearing after v are in M, we reverse σ' and then permute v and the first vertex. In both cases v becomes the first vertex of σ'.

It remains to consider the case when there are two vertices $a, b \notin M$, such that $a < v < b$ in σ'. Then there is a path from a to b in G, such that all vertices of the path are in $V \setminus M$. Consequently there are two consecutive vertices of the path, say a' and b', such that $a' < v < b'$ in σ'. Thus $\{v, a'\}$ and $\{v, b'\}$ are edges of H'. Since $a', b' \notin M$, by construction of H' we must have $a', b' \in S$. Recall that S is a minimal separator, thus there are two connected components C and D of $G - S$ such that $N(C) = N(D) = S$. At least one of them, say C, is different from M. Let μ be a path from a' to b' in $G[C \cup \{a', b'\}]$, not using the edge $\{a', b'\}$. Like above, there are two consecutive vertices a'' and b'' of μ with $a'' < v < b''$ in σ. Hence v is adjacent in H' to both a'' and b''. At least one of a'', b'' is in C, contradicting the fact that H' has no edges between v and $V \setminus (M \cup S)$. □

3.3.2 A family of nice layouts

Now we proceed to the description of an incremental step. It is built upon a partition of the vertex set induced by the prefix ρ computed so far. The parts are defined as follows.

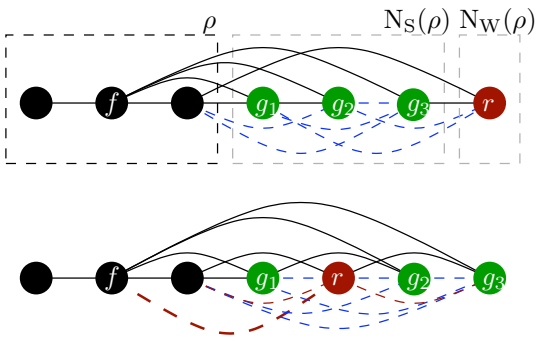


Figure 3.4: The strong neighborhood of ρ should come first.

Definition 3.3.2. *Let ρ be a non empty prefix. We denote by* $\mathrm{First}(\rho)$ *the first vertex in ρ having a neighbor in $V \setminus \mathrm{V}(\rho)$. We define the strong neighborhood (denoted* $\mathrm{N_S}(\rho)$*), weak neighborhood (*$\mathrm{N_W}(\rho)$*) and non-neighborhood* $\overline{\mathrm{N}}(\rho)$ *as follows:*

- $\mathrm{N_S}(\rho) = N(\mathrm{First}(\rho)) \setminus \mathrm{V}(\rho)$,
- $\mathrm{N_W}(\rho) = N(\mathrm{V}(\rho)) \setminus N(\mathrm{First}(\rho))$,
- $\overline{\mathrm{N}}(\rho) = V \setminus N[\mathrm{V}(\rho)]$.

In other words, the strong neighborhood of ρ are the grey neighbors of $\mathrm{First}(\rho)$. The weak neighborhood of ρ are the grey vertices which are not neighbors of $\mathrm{First}(\rho)$. The non-neighborhood of ρ are the white vertices.

It is a straightforward observation that in any BFS ordering σ, for any prefix ρ, the vertices of strong, weak and non-neighborhood of ρ are consecutive in σ and they appear exactly in this order. We formalize this condition with the following definition. Let us remind that the operator $\bullet$ denotes concatenation.

Definition 3.3.3. *We say that a layout σ* respects *a prefix ρ if σ is a refinement of $\rho \bullet \mathrm{N_S}(\rho) \bullet \mathrm{N_W}(\rho) \bullet \overline{\mathrm{N}}(\rho)$.*

Our goal is to show that if ρ is a nice prefix starting with a moplexian vertex, then there is a nice layout respecting it. This is a first step towards the extension of a nice prefix by adding a new vertex.

The next lemma shows that, given a prefix ρ, a layout σ with the strong neighborhood of ρ put right after ρ has an advantage over any layout σ' starting with ρ but without this property. The completion $\mathtt{PIG}(G, \sigma')$ adds a fill edge which is preserved as a non-edge in $\mathtt{PIG}(G, \sigma)$. In other words, there is a nice layout which is an extension of ρ with the strong neighborhood of ρ put right after ρ.

In Figure 3.4 there is an example showing in the top part how ρ is followed by $\mathrm{N_S}(\rho)$. In the bottom part there is a sample layout which does not follow this rule. Notice the fill edge e joining the vertex $\mathrm{First}(\rho)$, labeled f, and the vertex r which is not a neighbor of $\mathrm{First}(\rho)$. Such a fill edge does not appear if $\mathrm{N_S}(\rho)$ is put right after ρ in σ.

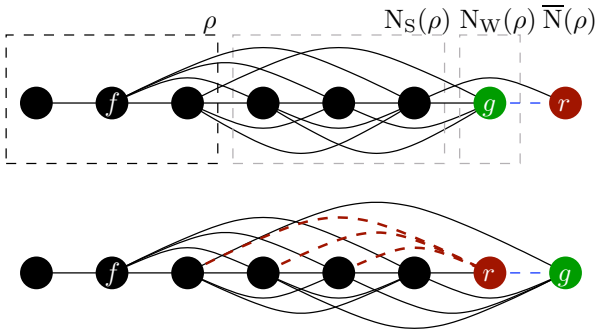


Figure 3.5: The weak neighborhood comes before the non-neighborhood.

Lemma 3.3.4. *Let σ and σ' be two layouts with a common prefix ρ and such that $\mathtt{PIG}(G,\sigma') \subseteq \mathtt{PIG}(G,\sigma)$. Suppose that σ is a refinement of $\rho \bullet \mathrm{N_S}(\rho) \bullet (\mathrm{N_W}(\rho) \cup \overline{\mathrm{N}}(\rho))$. Then σ' is also a refinement of $\rho \bullet \mathrm{N_S}(\rho) \bullet (\mathrm{N_W}(\rho) \cup \overline{\mathrm{N}}(\rho))$.*

Proof. Assume that both sets $\mathrm{N_S}(\rho)$ and $\mathrm{N_W}(\rho) \cup \overline{\mathrm{N}}(\rho)$ are not empty, otherwise the conclusion is true for any σ' starting with ρ. Let v_f denote $\mathrm{First}(\rho)$.

By contradiction suppose that there are two vertices $a \in \mathrm{N_S}(\rho)$ and $b \in \mathrm{N_W}(\rho) \cup \overline{\mathrm{N}}(\rho)$ such that $v_f < b < a$ in the layout σ'. Therefore $\{v_f, b\}$ is an edge of $\mathtt{PIG}(G,\sigma')$. If $\mathtt{PIG}(G,\sigma)$ contained the edge $\{v_f, b\}$, then there are two adjacent vertices v' and a' of G such that $v' \leq v_f < b \leq a'$ in the layout σ. By definition of $v_f = \mathrm{First}(\rho)$ we must have $v' = v_f$. Therefore $a' \in \mathrm{N_S}(\rho)$, contradicting the fact that $\mathrm{N_S}(\rho)$ appears before b in σ.

We conclude that the edge $\{v_f, b\}$ appears in $\mathtt{PIG}(G,\sigma')$ but not in $\mathtt{PIG}(G,\sigma)$. □

Suppose that σ is a layout respecting a prefix ρ and that σ' is another layout starting with ρ. By the previous lemma, we already know that if the completion $\mathtt{PIG}(G,\sigma')$ is included in $\mathtt{PIG}(G,\sigma)$, then the strong neighborhood of ρ must be right after ρ in σ'. The following lemma states that also the weak neighborhood of ρ must appear before the non-neighborhood of ρ. Putting things together, σ' has to respect ρ as well.

Lemma 3.3.5. *Let σ and σ' be two layouts with a common prefix ρ and such that $\mathtt{PIG}(G,\sigma') \subseteq \mathtt{PIG}(G,\sigma)$. Assume that σ respects ρ and let $u \in \mathrm{N_W}(\rho)$, $w \in \overline{\mathrm{N}}(\rho)$. Then u appears before w in σ'.*

Proof. By contradiction, suppose that w appears before u in σ'. Let $u' \in \mathrm{V}(\rho)$ be a neighbor of u. The edge $\{w, u'\}$ is present in $\mathtt{PIG}(G,\sigma')$, since w is between u' and u in σ'. On the other hand, σ respects ρ, so w appears after $\rho \bullet \mathrm{N_S}(\rho) \bullet \mathrm{N_W}(\rho))$. No element of ρ is adjacent in G to a vertex appearing after w in σ. By construction of $\mathtt{PIG}(G,\sigma)$, this graph does not contain the edge $\{w, u'\}$. □

In Figure 3.5 there is an example showing in the top part that the weak neighborhood is put before the non-neighborhood of ρ. In the bottom part there is a sample layout which does not follow this rule. Notice the fill edges joining the vertex r of the non-neighborhood of ρ with all white or gray neighbors of the vertex g. Such fill edges do not appear if g is put before r.

So Lemmas 3.3.4 and 3.3.5 directly imply the following:

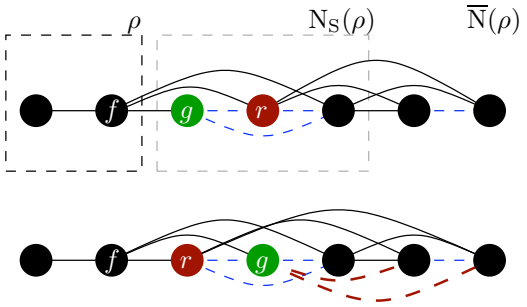


Figure 3.6: A vertex in $\mathrm{N_S}(\rho)$ having few white neighbors should come first.

Proposition 3.3.6. *Let σ and σ' be two layouts with a common prefix ρ and such that $\mathtt{PIG}(G,\sigma') \subseteq \mathtt{PIG}(G,\sigma)$. If σ respects ρ, then σ' also respects ρ.*

Having the BFS sufficient condition completed, now we have to focus on the tie-break rule of the algorithm. Given a prefix ρ, we have to choose between layouts that respect ρ. It is important to notice, that for such a layout σ the vertices after $\mathrm{First}(\rho)$ in ρ together with the strong neighborhood of ρ form a clique in $\mathtt{PIG}(G,\sigma)$. Moreover, $\mathrm{N_S}(\rho)$ together with $\mathrm{N_W}(\rho)$ form a clique in $\mathtt{PIG}(G,\sigma)$ as well. The reason is always that there is some edge of G that "covers" the corresponding vertices in the layout. Taking another permutation of vertices of $\mathrm{N_S}(\rho)$ makes no difference here. The only thing that may change are the fill edges incident to vertices in the non-neighborhood of ρ. This is the basis of the tie-break rule.

The following lemma states that, given a prefix ρ, it is safe to chose a vertex $u \in \mathrm{N_S}(\rho)$ with an inclusion minimal set of white neighbors to be the next in the layout.

Lemma 3.3.7. *Let ρ be a non-empty prefix. Let $u, w \in \mathrm{N_S}(\rho)$. Let σ be a layout that respects $\rho \bullet u$. Let σ' be a layout, with ρ as a prefix, in which w appears before u. If there is $w' \in (N(w) \cap \overline{\mathrm{N}}(\rho)) \setminus (N(u) \cap \overline{\mathrm{N}}(\rho))$, then the graph $\mathtt{PIG}(G,\sigma')$ contains an edge not appearing in $\mathtt{PIG}(G,\sigma)$.*

Proof. If w' is between $\mathrm{First}(\rho)$ and u in σ', then by Definition 3.1.1 $\{u, w'\}$ is present in $\mathtt{PIG}(G,\sigma')$. Else, u is between w and w' in σ' and the same holds. On the other hand, σ respects $\rho \bullet u$, so w' appears after $\rho \bullet u \bullet \mathrm{N_S}(\rho \bullet u) \bullet \mathrm{N_W}(\rho \bullet u))$. No element of $\rho \bullet u$ is adjacent in G to a vertex appearing after w' in σ. By Definition 3.1.1, $\{u, w'\}$ is not an edge of $\mathtt{PIG}(G,\sigma)$. □

In Figure 3.6 there is an example showing in the top part how ρ is followed by a vertex in $\mathrm{N_S}(\rho)$ having inclusion minimal set of white neighbors. In the bottom part there is a sample layout which does not follow this rule. Notice the fill edges joining the vertex labeled "g" with the white neighbors of the vertex labeled "r". Such fill edges do not appear in any completion which follows our rules.

This ends the series of lemmas justifying the rules of our algorithm. We can proceed to the main result of this chapter.

3.3.3 Nice layouts : a sufficient condition

Putting our rules together we can state the main theorem of this chapter which defines a family of nice layouts.

Theorem 3.3.8. *Let $G = (V, E)$ be a graph. Let $\sigma = (v_1, \ldots, v_n)$ be a layout of G such that v_1 is a moplexian vertex and for each $1 < i < n$:*

1. *σ respects ρ, where $\rho = (v_1, \ldots, v_{i-1})$,*
2. *v_i is such that $N(v_i) \cap \overline{\mathrm{N}}(\rho)$ is inclusion-minimal over all vertices in $\mathrm{N_S}(\rho)$.*

Then σ is a nice layout.

Proof. Suppose that σ is not a nice layout and let σ' be a layout such that $\mathtt{PIG}(G, \sigma')$ is a strict subgraph of $\mathtt{PIG}(G, \sigma)$. Take σ' in order to maximize the common prefix of σ and σ'. Let $\rho = (v_1, \ldots, v_p)$ be this maximum common prefix. By construction of σ, all the edges of $\mathtt{PIG}(G, \sigma)$ incident to v_1 are also edges of G. By Proposition 3.3.1, σ' starts with v. Consequently ρ has at least one vertex.

Let $u = v_{p+1}$ be the vertex of index $p+1$ in σ and w be the vertex of index $p+1$ in σ'.

By Proposition 3.3.6, σ' respects ρ.

Let σ'' be the layout obtained from σ' by exchanging u and w. We claim that $\mathtt{PIG}(G, \sigma') = \mathtt{PIG}(G, \sigma'')$. By definition of $\mathtt{PIG}(G, \sigma')$ and $\mathtt{PIG}(G, \sigma'')$, every edge in the symmetric difference $E(\mathtt{PIG}(G, \sigma'))\overline{\cup}E(\mathtt{PIG}(G, \sigma''))$ is incident to a vertex between u and w in σ'. Let I denote this interval. Since σ and σ' respect ρ, we have that $u, w \in \mathrm{N_S}(\rho)$, hence $I \subseteq \mathrm{N_S}(\rho)$. As a consequence of Lemma 3.3.7 and by the condition 2 of the theorem, $N(x) \cap \overline{\mathrm{N}}(\rho) = N(u) \cap \overline{\mathrm{N}}(\rho)$ for every $x \in I$. Let z be the last vertex of σ' contained in $N(u) \cap \overline{\mathrm{N}}(\rho)$, if such a vertex exists. In particular z is also the last vertex of σ'' in $N(u) \cap \overline{\mathrm{N}}(\rho)$.

Consider any $y \in \mathrm{V}(I)$. Both in $\mathtt{PIG}(G, \sigma')$ and $\mathtt{PIG}(G, \sigma'')$, y is adjacent to all vertices of $\mathrm{V}(\rho)$ appearing after $\mathrm{First}(\rho)$ and has no neighbor appearing strictly before $\mathrm{First}(\rho)$. Since $y \in \mathrm{N_S}(\rho)$, the vertices of $\overline{\mathrm{N}}(\rho)$ adjacent to y in $\mathtt{PIG}(G, \sigma')$ are precisely the ones appearing before z – or this neighborhood is empty if z does not exist. The same holds for $\mathtt{PIG}(G, \sigma'')$. Eventually, $\mathrm{N_S}(\rho) \cup \mathrm{N_W}(\rho)$ induces a clique both in $\mathtt{PIG}(G, \sigma')$ and $\mathtt{PIG}(G, \sigma'')$. Indeed the last vertex b of $\mathrm{N_S}(\rho) \cup \mathrm{N_W}(\rho)$ in σ' (resp. σ'') is adjacent in G to some vertex a of ρ. Since σ' and σ'' respect ρ, all the vertices of $\mathrm{N_S}(\rho) \cup \mathrm{N_W}(\rho)$ are in between a and b, so they form a clique. That proves that $\mathtt{PIG}(G, \sigma') = \mathtt{PIG}(G, \sigma'')$.

We have proved that σ'' and σ have $\rho \bullet u$ as common prefix, and $\mathtt{PIG}(G, \sigma'') \subseteq \mathtt{PIG}(G, \sigma)$. This contradicts the choice of σ'.

□

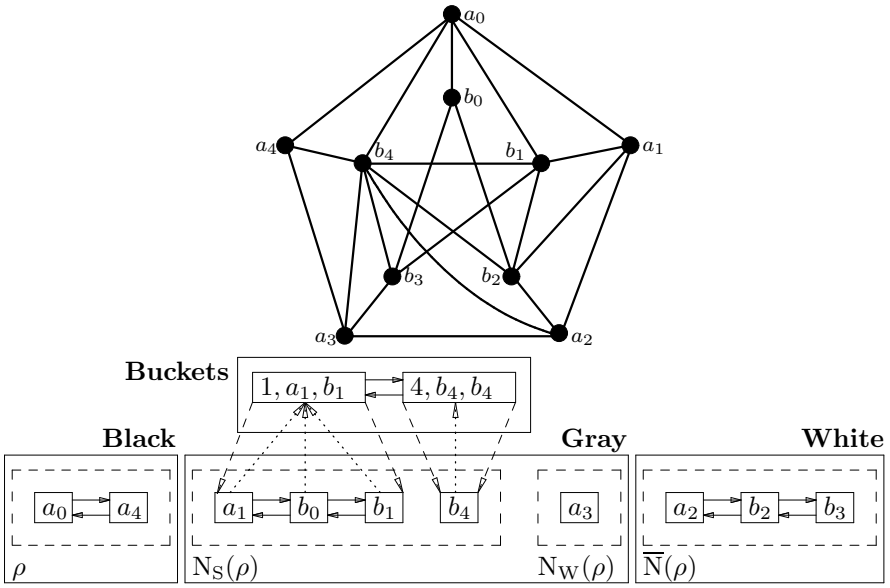
3.4 The algorithm

In this section we give an algorithm based on Theorem 3.3.8 which takes as input a simple graph G and computes in linear time a proper interval model.

The algorithm is based on BFS, see Figure 3.7. It creates a layout σ like in Theorem 3.3.8 and then returns a proper interval model of the minimal proper interval completion $\mathtt{PIG}(G, \sigma)$.

Theorem 3.4.1. *There is a linear time algorithm that, given an arbitrary graph G, computes a proper interval model of a minimal proper interval completion of G.*

Proof. The layout produced by the algorithm respects the conditions of Theorem 3.3.8. Indeed, it is sufficient to notice that the function **ChooseNextVertex** chooses a vertex in $\mathrm{N_S}(\rho)$ for the current prefix ρ, and moreover this vertex v is of minimum $\mathrm{d}_{\overline{\mathrm{N}}}(v)$. $\mathrm{d}_{\overline{\mathrm{N}}}(v)$ is the cardinality of



Algorithm MinimalProperIntervalCompletion

Input: a simple graph $G = (V, E)$;
Output: the proper interval model of a minimal proper interval completion of G;
Data structures:

$mark$: each vertex is marked white, grey or black (BFS color code).
Q is the queue of processed vertices.
ρ is the current prefix (an ordering on the black vertices).
$\mathrm{d}_{\overline{\mathrm{N}}}(v)$ is the number of white neighbors of the vertex v.

Function ChooseNextVertex : chooses a vertex v in the queue Q such that $v \in \mathrm{N_S}(\rho)$ and $\mathrm{d}_{\overline{\mathrm{N}}}(v)$ is minimum for this property.
Function IntervalModel : computes the interval model.
begin
compute a moplexian vertex v_1 and put $\rho := (v_1)$
mark all vertices as white
init Q with the neighbors of v_1 in G and mark these vertices as grey
mark v_1 as black
compute $\mathrm{d}_{\overline{\mathrm{N}}}(x)$ for all vertices x
for $i := 2$ to n do
  $v_i :=$ **ChooseNextVertex**()
  mark v_i as black, $\rho := \rho \bullet v_i$
  compute the set N_i of white neighbors of v_i
  mark the elements of N_i as grey and add them to Q
  for each $y \in N_i$ **for** each neighbor z of y $\mathrm{d}_{\overline{\mathrm{N}}}(z) := \mathrm{d}_{\overline{\mathrm{N}}}(z) - 1$
IntervalModel(σ)
end

Figure 3.7: Algorithm Minimal Proper Interval Completion and data structure.

Function IntervalModel

Input: $\sigma = (v_1, \ldots, v_n)$ - a BFS ordering of a simple connected graph G;
Output: an interval model of the graph $\texttt{PIG}(G, \sigma)$;
Data structures:
r is the biggest index of a neighbor of a vertex considered so far.
c is a counter for numbering the maximal cliques of $\texttt{PIG}(G, \sigma)$.
$[v_{l_c} : v_{r_c}]$, $1 \leq c \leq j$ are maximal cliques of $\texttt{PIG}(G, \sigma)$. (see Remark 2.2.13)
v_{l_c}, v_{r_c}, $1 \leq c \leq j$ are marks on the leftmost and rightmost vertices of c.
Q is a queue containing the cliques that the current vertex belongs to.

begin
$r := 1$
$c := 1$
for $i := 1$ **to** n **do**
  if $\max\{q \mid v_q \in N_G(v_i)\} > r$ **then**
    $r := \max\{q \mid v_q \in N_G(v_i)\}$
    mark v_i as v_{l_c}
    mark v_r as v_{r_c}
    increment c
for $i := 1$ **to** n **do**
  if v_i is marked as v_{l_c} **then**
    add c at the end of the queue Q
  assign to v_i the interval $[First(Q) : Last(Q)]$
  if v_i is marked as v_{r_c} **then**
    remove c from the beginning of the queue Q
$im :=$ the interval model
end

Figure 3.8: Algorithm Interval Model.

$N(v) \cap \overline{\mathrm{N}}(\rho)$, so it implies that the latter is inclusion-minimal over all white neighborhoods of the elements of $\mathrm{N_S}(\rho)$.

Let us discuss a linear time implementation of the algorithm. The choice of the first vertex can be done in linear time by Proposition 3.3.1. The main difficulty is that the function **ChooseNextVertex** must work in constant time. For this purpose, the queue Q will actually be a queue of sets $(N_{j_1}, N_{j_2}, \ldots, N_{j_k})$, where N_{j_p} is the set of neighbors of v_{j_p} added to the queue when processing v_{j_p} (empty sets are not enqueued). Hence $\mathrm{N_S}(\rho)$ is the first set in the queue, and vertices are dequeued from it.

Notice that it is not necessary to directly compare the sets of white neighbors to get an inclusion minimal one. It is enough to analyze their cardinalities. For this purpose, we introduce the notation $\mathrm{d}_{\overline{\mathrm{N}}}(v)$ to denote the cardinality of $|N(v) \cap \overline{\mathrm{N}}(\rho)|$. In order to choose the vertex $v \in \mathrm{N_S}(\rho) = N_{j_1}$ of minimum $\mathrm{d}_{\overline{\mathrm{N}}}(v)$ in constant time, we need to sort $\mathrm{N_S}(\rho)$ by increasing $\mathrm{d}_{\overline{\mathrm{N}}}()$. Notice that the value of some $\mathrm{d}_{\overline{\mathrm{N}}}(z)$ might change during the algorithm, and we wish to update the value in constant time. We use a bucket sort with a special data structure (see [67, 65] for a detailed description of the data structure, the authors use it for partition refinement algorithms). The buckets are kept

as a doubly chained list (instead of the usual array). Each bucket has its value (the $\mathrm{d}_{\overline{\mathrm{N}}}(u)$ for the elements of the bucket), and points towards the previous and next non-empty buckets, according to their values. The vertices of a bucket are kept in a doubly chained list, and each vertex points towards the bucket to which it belongs. An example is given in Figure 3.7, where the bucket of value 1 contains a_1, b_0, b_1 and the bucket 4 contains b_4.

When a set N_i becomes the first set of Q, we apply a classical bucket sort on the vertices of N_i. This sort costs $\mathcal{O}(|N_i| + \max\{\mathrm{d}_{\overline{\mathrm{N}}}(u) \mid u \in N_i\})$. Then we construct our data structure for the buckets, within the same running time. During the whole algorithm, this initialization of the buckets costs $\mathcal{O}(n+m)$, due to the fact that the sets N_i are pairwise disjoint.

During the algorithm, we decrement the value $\mathrm{d}_{\overline{\mathrm{N}}}(z)$ for some vertices z (see the two last **for** loops). If z is in the set $\mathrm{N_S}(\rho)$, we must update the buckets in constant time. Let B be the bucket containing z and B' be the previous bucket in the list of buckets. If the bucket B' corresponds to the value $\mathrm{d}_{\overline{\mathrm{N}}}(z) - 1$ (before decrementing it), we simply move z from B to B', and possibly remove B if it becomes empty. Otherwise, B' corresponds to a value strictly smaller than $\mathrm{d}_{\overline{\mathrm{N}}}(z) - 1$, we create a new bucket B'', of value $\mathrm{d}_{\overline{\mathrm{N}}}(z) - 1$, and add it to the list of buckets between B' and B. Thanks to our data structure, this operation can be done in linear time. Note that the total number of iterations of the two last **for** loops is at most $n+m$. Indeed, each vertex y becomes grey exactly once, thus each edge $\{y, z\}$ is visited at most twice.

The function **IntervalModel** (see Figure 3.8) constructs a clique path of $\mathtt{PIG}(G, \sigma)$ like in Remark 2.2.13 and computes an interval model based on this clique path in linear time. We only need to prove that a maximal clique path of a proper interval graph gives indeed a proper interval model as in Definition 2.2.4.

Suppose on the contrary that there are two vertices u, v such that the interval P_v of P of maximal cliques containing v is properly contained in the interval P_u of maximal cliques containing u. Therefore, both endpoints of P_u do not intersect P_v. Let a be a vertex contained in the left endpoint K_L of P_u and in no other clique of P_u. It exists, since K_L is a maximal clique of G. Analogously, let z be a vertex contained in K_R and in no other clique of P_u. It is easy to verify that $\{a, u, v, b\}$ induces a $K_{1,3}$. This contradicts the characterization of proper interval graphs by Theorem 2.2.26.

□

3.5 Conclusions

We presented a linear time algorithm computing a minimal proper interval completion of an arbitrary graph. It is a standard BFS exploration algorithm with only a simple additional tie-break rule based on a parameter that can be computed for each vertex locally.

There are two very natural questions related to minimal proper interval completions that we leave open. The first would be to characterize all minimal proper interval completions, for example by describing all the layouts σ such that $\mathtt{PIG}(G, \sigma)$ is a minimal proper interval completion of G. We point out that our algorithm cannot obtain all such layouts of the input graph. Indeed, if we consider the graph $K_{1,4}$, our algorithm chooses a simplicial vertex and completes the rest into a clique. A different minimal proper interval completion of the $K_{1,4}$ can be obtained by adding a matching to the independent set.

The second question consists in extracting a minimal proper interval completion from some non-minimal proper interval completion H of G. The naive technique would consist in checking,

for each edge $e \in E(H) \setminus E(G)$, if $H - e$ is a proper interval graph. Although this idea works for minimal triangulations and minimal split completions, in our case we have examples showing that it does not always yield a minimal proper interval completion.

Finally, layouts in the context of proper interval graphs make a natural framework for tackling bandwidth. First, because bandwidth is defined as the minimum of $\max_{j,k \in 1..n} |j - k|, v_j v_k \in E$, over all layouts of G [29]. Then again, because an equivalent definition of bandwidth states that it is the minimum clique number over proper interval embeddings of G. Finally, given a prefix ρ, our algorithm puts the neighborhood of First(ρ) right after rho. This may be regarded as a naive attempt to compute the bandwidth, by minimizing the "span" of the edges incident to First(ρ). A more sophisticated technique of verifying if the bandwidth of G is at most k would consist in using our algorithm with a modified definition of the strong neighborhood of ρ. For this purpose we need to define $Bound(\rho)$, the first vertex in ρ, such that the grey neighbors of vertices between First(ρ) and $Bound(\rho)$ in ρ have to be placed right after ρ in order for the maximum span of the incident edges not to be greater than k. We have verified that taking this set as the new definition of $\mathrm{N_S}(\rho)$ yields an algorithm that emulates the result of [82] computing bandwidth of interval graphs. So it is an interesting question if the modified algorithm has some interesting properties for other graph classes.

Chapter 4

Minimal Interval Completion Through Exploration

Contents

Encouraged by the successful attempt to compute a minimal proper interval completion with a simple exploration algorithm, we asked the natural question if it was possible to obtain similar results for minimal interval completion. The characterization of proper interval graphs by bicompatible orderings, given in Theorem 2.2.14, is a very strong tool. During construction of a layout σ of G to be used to compute the completion H it defines, any fixed prefix already tells a lot about the structure of H. Any edge of G defines a clique of H on the set of vertices it "covers" in σ (see Definition 3.1.1). An interval ordering, given in Theorem 2.2.12, gives much more freedom for minimal interval completions. With more possibilities to choose among, it is more difficult to ensure that the computed interval completion is minimal. Even though the tools we initially had were not strong enough, we managed to build them up to match the needs, without a big increase in complexity. We give a $\mathcal{O}(nm)$ time complexity algorithm computing a minimal interval completion of an arbitrary graph.

4.1 Interval layout

First we need to define the new completion operator that works with the interval ordering characterization of interval graphs given in Theorem 2.2.12 (see Subsection 2.2.3).

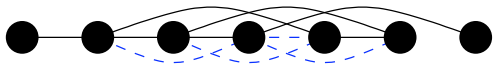


Figure 4.1: $\texttt{IG}(G, \sigma)$ - dashed fill edges.

Definition 4.1.1. *Let $G = (V, E)$ be an arbitrary graph and $\sigma = (v_1, \ldots, v_n)$ be an ordering of V. The graph $\mathit{IG}(G, \sigma) = (V, E')$ is defined by*

$$E' = \{\{v_i, v_k\} \mid \exists j : i < k \leq j, v_i v_j \in E\}.$$

This operation corresponds to taking, for each edge e of G, the first vertex l_e in σ incident to e and making it adjacent to every vertex between l_e and r_e in σ, where r_e is the last vertex in σ incident to e.

Analogously to the PIG operator and a proper interval completion, by Theorem 2.2.12, $\texttt{IG}(G, \sigma)$ yields an interval completion of G.

Lemma 4.1.2. *$\mathit{IG}(G, \sigma)$ is an interval graph.*

And also here we can prove that any minimal interval completion can be obtained by the operator IG on some layout of G.

Theorem 4.1.3. *Let $G = (V, E)$ be an arbitrary graph and $G' = (V, E')$ be a minimal interval completion of G. Then there is a layout σ such that $G' = \mathit{IG}(G, \sigma)$.*

Proof. By Theorem 2.2.12, there is a layout σ of G such that $G' = \texttt{IG}(G, \sigma)$. As a straight consequence of Definition 4.1.1, $E(\texttt{IG}(G, \sigma)) \subseteq E(G')$. By Lemma 4.1.2, $\texttt{IG}(G, \sigma)$ is also an interval graph. Thus, by minimality of G', we deduce that $E(\texttt{IG}(G, \sigma)) = E(G')$. □

To simplify the description, let us slightly abuse the notation and reuse the notion of "nice" in the context of minimal interval completions.

Definition 4.1.4. *An layout $\sigma = (v_1, \ldots, v_n)$ is called* nice *if $\mathit{IG}(G, \sigma)$ is a minimal interval completion of G. Any prefix $(v_1, \ldots, v_k)$, $k \leq n$ of an nice ordering is called an* nice prefix*.*

4.2 Nice layouts and nice prefixes

The minimal interval completion algorithm presented in this chapter is in many ways similar to the one for minimal proper interval completions presented in Chapter 3. It starts with a LexBFS-terminal vertex as a nice prefix and incrementally extends it, adding well chosen vertices one by one. The prefix under construction is at each step nice and eventually a nice layout is obtained. Although the presented algorithm is a BFS exploration with a particular tie-break rule, the characterization of nice layouts we give is more general.

In contrast with minimal proper interval completions, a moplexian vertex does not always constitute a nice prefix in the sense of minimal interval completions. We need more to ensure this property. Fortunately, we have found that the LexBFS again comes handy, since a LexBFS-terminal vertex carries enough structure.

4.2.1 LexBFS-terminal vertex

Recall that a vertex v numbered 1 by some execution of LexBFS is called a *LexBFS-terminal vertex*. A moplex M such that some execution of LexBFS terminates on a vertex of M is called a *LexBFS-terminal moplex*.

Berry and Bordat discovered that not only a LexBFS terminates on a moplexian vertex v_1 of G, but the order in which it explores G is very particular with respect to the moplex M to which v_1 belongs. Let $S = N_G(M)$, recall that S is a minimal separator of G. The connected components $\mathcal{C}(S) = \{C_1, \ldots, C_k\}$, are explored in ascending order with respect to inclusion of their neighborhoods. Moreover, each vertex in the neighborhood of a component is universal in the closed neighborhoods of the components explored later. The important fact which we use later is that, thanks to this property, the sequence of blocks $(O_1, \ldots, O_k)$, where $O_i = C_i \cup N_G(C_i)$, is a path decomposition of G (see Definition 2.2.9).

Lemma 4.2.1 ([7, 8])**.** *Let M be a LexBFS-terminal moplex and $S = N_G(M)$. Denote by $C_1, C_2, \ldots, C_k$, with $C_k = M$, the connected components of $G - S$ in the order in which the LexBFS execution encounters them. Then the following equation are satisfied:*

$$N(C_1) \subseteq N(C_2) \subseteq \cdots \subseteq N(C_k). \tag{4.1}$$

$$\begin{gathered}\forall i, j, x, y : 1 \leq i < j \leq k, x \in N(C_i), y \in N[C_j] \setminus N(C_i) \\ \Rightarrow \{x, y\} \in E(G).\end{gathered} \tag{4.2}$$

4.2.2 Choosing a first vertex

We want to prove that a LexBFS-terminal vertex v_1 constitutes a nice prefix. Thus, we will show that there is a nice layout σ that starts with v_1, and that for any minimal interval completion that does not have an interval ordering starting with v_1 there is a fill edge e incident to v_1 which is not present in $\texttt{IG}(G, \sigma)$. This follows the line of argumentation by which obeying our rules preserves a non-edge in the completion defined by the constructed layout which otherwise becomes a fill edge.

Lemma 4.2.2 does not speak of a LexBFS-terminal vertex directly, because the conditions expressed in Equations 4.1 and 4.2 are more general, and let us apply the lemma in more possible situations. Recall that the notions of a clique path and an interval ordering are interrelated by Remark 2.2.13.

Lemma 4.2.2. *Consider a non-complete graph $G = (V, E)$. Let v be a vertex of a moplex M and $S = N_G(M)$. Let $C_1, C_2, \ldots, C_k$, with $C_k = M$, the connected components of $G - S$, satisfy Equations 4.1 and 4.2 of Lemma 4.2.1. Then there exists a minimal interval completion H of G such that $N_G(v) = N_H(v)$.*

For any such H, there exists a clique path P of H such that $M \cup S$ is one of its end cliques.

Proof. Let H' be the graph obtained from G by transforming $N_G[C_i]$ into a clique, from each $1 \leq i \leq q$. By Equation 4.1 (see Lemma 4.2.1), $(N_G[C_1], \ldots, N_G[C_k])$ is a clique path of H', in particular H' is an interval graph. Consequently H' contains some minimal interval completion H of G as required.

Now let H be any minimal interval completion of G such that $N_H(v) = N_G(v)$. We first show that S induces a clique in H. Let D be a component of $G - S$, different from M, such that $N_G(D) = S$. Note that S is a v, u-minimal separator of G, for some $u \in D$. Let T be a minimal

v, u separator of H such that $T \subseteq N_H(v)$. Clearly T exists because u and v are non-adjacent in H. We claim that $S \subseteq T$. For each vertex $s \in S$, there is a u, v path of G contained in $D \cup \{v, s\}$. This path intersects $N_G(v)$ only in s, so also in the graph H the only possible intersection between T and the path is s. It follows that $s \in T$, so $S \subseteq T$. The minimal separator T induces a clique in H by Lemma 2.2.20. Hence S also induces a clique in H. Note that, by definition of a moplex, $M \cup S$ also induces a clique in H.

For each i, $1 \leq i \leq k$ let $H_i = H[N_G[C_i]]$. Let H'' be the graph with vertex set V and edge set $E(H_1) \cup E(H_2) \cup \cdots \cup E(H_k)$. Therefore $G \subseteq H'' \subseteq H$. We will construct a clique path P of H'', showing that H'' is an interval graph. By minimality of H, this implies that $H'' = H$. Moreover, the clique path P will have $M \cup S = N_G[M]$ as one of its end cliques.

Let $S_i = N_G(C_i)$. By Equation 4.2, the vertices of S_{i-1} are adjacent to all vertices of $C_i - S_{i-1}$ in the graph G, so also in H_i. Combined with the fact that $S_{i-1} \subseteq S$ induces a clique in H we have that S_{i-1} is contained in each maximal clique of H_i. We claim that for each i, $1 \leq i < k$, there exists a clique path of H_i such that S_i is contained in the rightmost clique of P_i. Indeed, the graph $H_i^+ = H[C_i \cup S \cup M]$ is an interval graph and $M \cup S$ is one of its maximal cliques. Take any clique path P_i^+ of H_i^+, we prove that $M \cup S$ is an end clique. By contradiction, let x (resp y) be a vertex appearing in the clique left (resp. right) to $S \cup M$, but not appearing in $S \cup M$. By the properties of a clique path, $S \cup M$ must separate x and y in H_i^+. This contradicts the fact that $x, y \in C_i$ and there exists an x, y-path in $G[C_i]$. So the only possibility is that $S \cup M$ is at an end of P_i^+. Since $S_i \subseteq S$ and every vertex of S has a neighbor in M, S_i is contained in the clique next to $S \cup M$ in P_i^+. The clique path P_i of H_i obtained by removing $S \cup M$ from P_i^+ has the required property. Eventually, by concatenating the clique paths $P_1, P_2, \ldots, P_k$, it is easy to check that we obtain a clique path P of H''. Indeed if a vertex x appears in the subpaths P_i and P_j with $i < j$, then $x \in N_G[C_i] \cap N_G[C_j] = S_i$ (see Equation 4.1). By Equation 4.2, x appears in every clique of P_k, for each $k, i < k \leq j$. Since H_k is the complete graph with vertex set $N_G[M] = S \cup M$, the clique path P has $S \cup M$ as rightmost clique. □

Thus we may prove that a LexBFS-terminal moplexian vertex is a good starting point for the construction of a nice layout.

Theorem 4.2.3. *Let G be a non-complete graph and v be a LexBFS-terminal moplexian vertex of G. For any minimal interval completion H of G such that $N_G(v) = N_H(v)$, there is an interval ordering of H starting with v.*

Proof. By Lemma 4.2.2, there exists a clique path of H such that the left-most clique is $M \cup S$, where $S = N(M)$. We can reverse this path so that $S \cup M$ becomes the leftmost clique of the clique path P. By construction, H has no fill edges incident to v, in particular the v only appears in the left-most clique of P. By Remark 2.2.13, there is an interval ordering of H starting with v. □

4.2.3 A family of nice layouts

Now we proceed to rules telling, given a prefix ρ, which vertex should be taken next. Here we are interested in only two subsets of $V \setminus \mathrm{V}(\rho)$. The set Nxt is an inclusion minimal set of vertices over the grey neighborhoods of vertices in ρ. The set Rst are the other vertices, that are neither in ρ nor in Nxt. At some point there may be several sets that fulfill the definition of Nxt, in such a situation any of them can be chosen arbitrarily. However, the algorithm we give is less general and chooses the set which is the grey neighborhood of the leftmost vertex in ρ, over the vertices that have non-empty grey neighborhood.

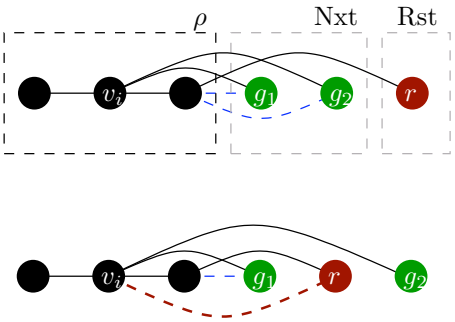


Figure 4.2: Putting Nxt right after ρ.

Definition 4.2.4. *We denote by $\rho = (v_1, \ldots, v_k)$ a prefix, and $R = V \setminus \mathrm{V}(\rho)$. Let* Nxt *be a non-empty subset of R, such that* $\mathrm{Nxt} = N_G(v_i) \cap R$ *for some $v_i \in \mathrm{V}(\rho)$ and* Nxt *is inclusion-minimal for this property. We denote $R \setminus \mathrm{Nxt}$ by* Rst*.*

The next lemma shows that, given a prefix ρ, we can fix a set Nxt according to Definition 4.2.4 and put it right after ρ in the constructed layout σ. It preserves some non-edges in the completion defined by σ that would otherwise become fill edges.

Lemma 4.2.5. *Let σ be a refinement of $\rho \bullet \mathrm{Nxt} \bullet \mathrm{Rst}$ and σ' be a refinement of $\rho \bullet R$ such that $\mathtt{IG}(G, \sigma') \subseteq \mathtt{IG}(G, \sigma)$. Then σ' also is a refinement of $\rho \bullet \mathrm{Nxt} \bullet \mathrm{Rst}$.*

Proof. Let $v_i \in \mathrm{V}(\rho)$ such that $\mathrm{Nxt} = R \cap N_G(v_i)$. Suppose that σ' is not a refinement of $\rho \bullet \mathrm{Nxt} \bullet \mathrm{Rst}$, so there is some vertex $u \in \mathrm{Rst}$ and a vertex $w \in \mathrm{Nxt}$ such that u appears before w in σ'. Since u appears in σ after all vertices of Nxt, v_i and u are not adjacent in $\mathtt{IG}(G, \sigma)$. Now in σ', u appears after v_i and before w. Since $\mathrm{Nxt} \subseteq N_G(v_i)$, v_i and w are adjacent in G and therefore v_i and u are adjacent in $\mathtt{IG}(G, \sigma')$ – a contradiction. □

In Figure 3.4 there is an example showing in the top part a layout σ in which ρ is followed by the neighborhood of v_i equal $\{g_1, g_2\}$. Notice that the grey neighborhood of v_i is inclusion minimal over all non-empty neighborhoods of vertices in ρ. In the bottom part, r appears before the last vertex of Nxt. For this reason, the fill edge $v_i r$ is present, which is preserved as a non-edge in the completion defined by the first layout.

In Lemma 4.2.5, there is no recursive assumption on shorter prefixes. So, in the example, we might as well have chosen the singleton neighborhood $\{r\}$ of the last vertex in ρ to be the set Nxt, since it also is inclusion minimal. In that case, the vertex r would have been put before g_1, g_2. This freedom is no longer present in the complete characterization of nice layouts. There, a layout σ should be a refinement of $\rho \bullet \mathrm{Nxt} \bullet \mathrm{Rst}$, for every ρ which is a prefix of σ. With this condition, g_1, g_2 have to appear before r.

The following lemma shows that, given a layout σ which is a refinement of $\rho \bullet \mathrm{Nxt} \bullet \mathrm{Rst}$ and the corresponding completion $H = \mathtt{IG}(G, \sigma)$, permuting vertices of Nxt can affect only edges with both incident vertices in Nxt. In this way, the permutation we choose for Nxt (and the interval completion of $G[\mathrm{Nxt}]$ which it defines) has no influence on the rest of the graph.

Lemma 4.2.6. *Consider two layouts σ and σ' of G that are refinements of $\rho \bullet \mathrm{Nxt} \bullet \sigma_{\mathrm{Rst}}$, where σ_{Rst} is an ordering of* Rst*. That is to say, σ and σ' differ only by a permutation of* Nxt*. Let u, v be two vertices adjacent in $\mathtt{IG}(G, \sigma')$ but non-adjacent in $\mathtt{IG}(G, \sigma)$. Then both $u, v \in \mathrm{Nxt}$.*

Proof. By construction of $\mathtt{IG}(G, \sigma)$ and $\mathtt{IG}(G, \sigma')$, at least one of the vertices u, v are in Nxt. By contradiction, suppose that the other is not in Nxt.

First we consider the case when $u \in \mathrm{V}(\rho)$ and $v \in \mathrm{Nxt}$. Suppose that u has a neighbor $u' \in \mathrm{Rst}$. In both $\mathtt{IG}(G, \sigma)$ and $\mathtt{IG}(G, \sigma')$ all vertices of Nxt are adjacent to u as they appear after u and before u' in the corresponding layout – a contradiction. So $N_G(u) \cap R \subseteq \mathrm{Nxt}$. By definition (minimality) of Nxt, either $\mathrm{Nxt} \subseteq N_G(u)$ or $\mathrm{Nxt} \cap N_G(u) = \emptyset$. Clearly, in the first case Nxt is contained in the neighborhood of u in both $\mathtt{IG}(G, \sigma)$ and $\mathtt{IG}(G, \sigma')$. In the second, for both $\mathtt{IG}(G, \sigma)$ and $\mathtt{IG}(G, \sigma')$ the vertex u has no neighbors in Nxt – a contradiction.

It remains to consider the situation when $u \in \mathrm{Nxt}$ and $v \in \mathrm{Rst}$. Since u and v are adjacent in $\mathtt{IG}(G, \sigma')$, there is a neighbor v' of u in G, appearing after v in σ'. But u, v, u' appear in the same order in σ, so u and v are adjacent in $\mathtt{IG}(G, \sigma)$ – a contradiction. □

4.2.4 Nice layouts : a sufficient condition

Nevertheless, the rest of the graph, and the set Rst in particular, has a big impact on which permutation of Nxt can be part of a nice layout. In fact, the task of choosing the next vertex from Nxt can be made independently of ρ, only taking care of the fact that Rst appears after Nxt in σ. For this reason, we can treat the task of ordering Nxt as the problem of finding a layout of Nxt that defines an inclusion minimal interval completion of $G[\mathrm{Nxt}]$, over completions that can be "glued" with an interval completion of $G[\mathrm{Rst}]$. This problem is, roughly speaking, equivalent to the original problem on an auxiliary graph which we define below. That is why, LexBFS comes handy as a tie-break rule for Nxt.

Definition 4.2.7. *Let σ be a layout of G, let ρ be a prefix of σ, and let* Nxt *be like in Definition 4.2.4. We denote by T the set of vertices of* Nxt *having neighbors in* Rst*. G_{Nxt} denotes the graph obtained from $G[\mathrm{Nxt}]$ by adding a dummy vertex d_1, adjacent to each vertex of T, and a dummy vertex d_2 adjacent only to d_1. The graph G^+_{Nxt} is obtained from G_{Nxt} by completing T into a clique.*

With this notations we may present the main result of this chapter. The techniques of the proof are an extension of the ones used in Theorem 3.3.8.

Theorem 4.2.8. *Let σ be a layout of G with the following properties:*

1. *σ starts with a LexBFS-terminal vertex v_1.*
2. *For any non-empty prefix $\rho = (v_1, \ldots, v_i)$*
   - *σ respects ρ, i.e. σ is a refinement of $\rho \bullet \mathrm{Nxt} \bullet \mathrm{Rst}$,*
   - *the next vertex in σ is a LexBFS-terminal vertex of G^+_{Nxt} obtained by running LexBFS starting from d_2.*

Then σ is a nice layout.

Proof. Suppose that $\sigma = (v_1, \ldots, v_n)$ is not nice and let σ' be a layout such that $H' = \mathtt{IG}(G, \sigma')$ is a minimal interval completion of G strictly contained in $H = \mathtt{IG}(G, \sigma)$. Take σ' such that the maximal common prefix ρ of σ and σ' is the longest possible.

Claim 1. *ρ is not empty.*

The first vertex v_1 of σ is LexBFS-terminal. $N_G(v_1) = N_{\mathtt{IG}(G,\sigma)}(v_1)$, since σ respects the prefix (v_1) and thus the neighbors of v_1 in G appear right after v_1 in σ. So the Claim follows by Theorem 4.2.3.

Let v (resp. u) be the vertex right after ρ in σ (resp. in σ'). By Lemma 4.2.5, we have:

Claim 2. *σ' is a refinement of $\rho \bullet \mathrm{Nxt} \bullet \mathrm{Rst}$, in particular $u \in \mathrm{Nxt}$.*

Claim 3. *Let σ'' be any refinement of $\rho \bullet \mathrm{Nxt} \bullet \mathrm{Rst}$ and $H'' = \mathtt{IG}(G, \sigma'')$. Let $P'' = P(G, \sigma'')$ (see Remark 2.2.13). Then $H''[\mathrm{Nxt}]$ is an interval completion of $G[\mathrm{Nxt}]$, where the clique path $P''[\mathrm{Nxt}]$ has the set $T = N_G(\mathrm{Rst}) \cap \mathrm{Nxt}$ contained in one of the end-cliques. In particular, T is a clique in H''.*

Clearly, $P^{''}[\mathrm{Nxt}]$ is a clique path of $H^{''}[\mathrm{Nxt}]$. The last clique contains T, since the corresponding intervals in the model intersect the interval of a vertex in Rst.

Claim 4. *$H'[\mathrm{Nxt}]$ is an interval completion of $G[\mathrm{Nxt}]$, minimal with respect to the property expressed in the previous claim.*

Since σ' defines a minimal interval completion H' of G, σ' has to yield $H'[\mathrm{Nxt}]$ minimal with this property. Suppose it is not minimal, and let $H^{'''}[\mathrm{Nxt}]$ be the corresponding completion strictly included in $H^{'}[\mathrm{Nxt}]$. Then we can take the corresponding clique path $P^{'''}[\mathrm{Nxt}]$ to create an interval order $\sigma^{'''}_{\mathrm{Nxt}}$ of $H^{'''}[\mathrm{Nxt}]$ (see Remark 2.2.13). By Lemma 4.2.6, $\sigma^{'''} = \rho \bullet \sigma^{'''}_{\mathrm{Nxt}} \bullet \sigma^{'}_{\mathrm{Rst}}$ yields $H^{'''} = G(\sigma^{'''})$ strictly contained in $H^{'}$. A contradiction with minimality of $H^{'}$.

Following Definition 4.2.7, let H'_{Nxt} be obtained from $H'[\mathrm{Nxt}]$ by adding a dummy vertex d_1 adjacent to the vertices of T and a vertex d_2 adjacent to d_1.

Claim 5. *H'_{Nxt} is a minimal interval completion of G^+_{Nxt}.*

Given a clique path P of H, let $P[\mathrm{Nxt}]$ denote the clique path of $H[\mathrm{Nxt}]$ obtained by restricting all the bags of P to their intersections with Nxt and then removing the redundant ones (leaving only unique maximal cliques of $H[\mathrm{Nxt}]$).

Let P'_{Nxt} denote the clique path of H'_{Nxt}, obtained from $P'[\mathrm{Nxt}]$ by adding two bags $q_1 = T \cup \{d_1\}$ and $q_2 = \{d_1, d_2\}$ after the clique containing T (see Claim 2). It is a clique path indeed, so H'_{Nxt} is an interval completion of G^+_{Nxt}. Suppose it is not minimal. So there is a minimal one $H^{''}_{\mathrm{Nxt}}$ strictly included in H'_{Nxt}. Notice that this graph has a clique path $P^{''}_{\mathrm{Nxt}}$, that also has $--q_1--q_2$ at an end. Indeed, the moplex $M = d_2$ satisfies the conditions of Lemma 4.2.2 in $H^{''}_{\mathrm{Nxt}}$, so there is a clique path of $H^{''}_{\mathrm{Nxt}}$ with $\{d_1, d_2\}$ as one of the end-cliques. Therefore $P^{''}[\mathrm{Nxt}]$, obtained by removing $--q_1--q_2$ from $P^{''}_{\mathrm{Nxt}}$, is a clique path of $H^{''}[\mathrm{Nxt}]$ with T contained in one of the end-cliques. Which contradicts Claim 3, since $H^{''}[\mathrm{Nxt}]$ is a strict subgraph of $H'[\mathrm{Nxt}]$.

Claim 6. *There is an interval ordering of H'_{Nxt} starting with v.*

Let us prove that $N_{H'_{\mathrm{Nxt}}}(v) = N_{G^+_{\mathrm{Nxt}}}(v)$. Indeed if $v \in T$, since v is the last vertex encountered by LexBFS launched on G^+_{Nxt} from d_2, we have $T = \mathrm{Nxt}$. In this case the neighborhood of v in both graphs is $T \setminus \{v\} \cup \{d_1\}$, and the equality follows.

Now if $v \notin T$ then $N_G(v) \cap R \subset \mathrm{Nxt}$. By second condition of the theorem, σ respects $\rho \bullet v$, so $N_G(v) \cap \mathrm{Nxt}$ is put before $R \setminus N_G(v)$ in σ and $N_{H[\mathrm{Nxt}]}(v) = N_{G[\mathrm{Nxt}]}(v)$. Therefore $N_{H'_{\mathrm{Nxt}}}(v) = N_{G^+_{\mathrm{Nxt}}}(v)$, since

$$N_{G^+_{\mathrm{Nxt}}}(v) \subseteq N_{H'_{\mathrm{Nxt}}}(v) \subseteq N_{H_{\mathrm{Nxt}}}(v) = N_{G_{\mathrm{Nxt}}}(v) \subseteq N_{G^+_{\mathrm{Nxt}}}(v).$$

The claim follows from Theorem 4.2.3 and Claim 4.

Claim 7. *There is a layout $\sigma^{''}$, with $G(\sigma^{''}) = \mathtt{IG}(G, \sigma')$, sharing a longer prefix with σ – a contradiction.*

We restrict the ordering from the previous claim to Nxt and obtain $\sigma^{''}_{\mathrm{Nxt}}$. Let $\sigma^{''} = \rho \bullet \sigma^{''}_{\mathrm{Nxt}} \bullet \sigma^{'}_{\mathrm{Rst}}$. By Lemma 4.2.6, $G(\sigma^{''}) = \mathtt{IG}(G, \sigma')$. So $\sigma^{''}$ defines the same completion and shares a longer prefix. Which contradicts the choice of $\sigma^{'}$.

This completes the proof of the theorem. □

4.3 The algorithm

In this section we give an algorithm that, given an arbitrary graph G, computes an interval model of a minimal interval completion H of G.

Theorem 4.3.1. *There is an $\mathcal{O}(nm)$-time algorithm computing a minimal interval completion of an arbitrary graph.*

Proof. We prove that the algorithm `MIC_Ordering` of Figure 4.3 computes in $\mathcal{O}(nm)$ time a layout satisfying the conditions of Theorem 4.2.8.

Clearly the first vertex v_1 is a LexBFS-terminal vertex, implying the first condition of Theorem 4.2.8. The initialization of the ordered partition ensures that all neighbors of v_1 in G appear contiguously and right after v_1 in σ. Therefore σ is a refinement of $v_1 \bullet N_G(v_1) \bullet V \setminus N_G[v_1]$.

The algorithm maintains an ordered partition of the vertex set of G. At each step i the set Nxt corresponds like in Definition 4.2.4 to the prefix $\rho = (v_1, \ldots, v_i)$ as required. By contradiction suppose there exists $j < i$ such that $N(v_j)$ is strictly contained in Nxt. Then at step j, the class C of the ordered partition containing Nxt has been split in $C \cap N_G(v_j)$ and $C \setminus N_G(v_j)$ – contradicting the fact that Nxt is a class of the ordered partition at the step i.

Unlike the Theorem 4.2.8, our algorithm computes the vertex v_i by launching LexBFS from d_2 on the graph G_{Nxt} and not G^+_{Nxt}. The reason is related to the running time. Indeed G_{Nxt} has $\mathcal{O}(n+m)$ edges, while if we compute G^+_{Nxt}, the number of edges of G^+_{Nxt} might be up to $\Omega(n^2)$. Nevertheless we prove that v_i is also a LexBFS-terminal vertex obtained by using G^+_{Nxt} instead of G_{Nxt}. Let n' be the number of vertices of G_{Nxt}. When the vertex d_1 is numbered (with number $n'-1$) by LexBFS on G_{Nxt}, all vertices of T are labeled $(n-1)$. Then all vertices of T are numbered before the vertices of $\mathrm{Nxt} \setminus T$. The same would have happened by running LexBFS on G^+_{Nxt}. Moreover, in G^+_{Nxt} any numbering of T is valid in this case. So the LexBFS numbering on G_{Nxt} is also a LexBFS numbering on G^+_{Nxt}.

Together with the way that algorithm maintains an ordered partition, it implies the second condition of Theorem 4.2.8.

Each iteration of the **for** loop must be performed in $\mathcal{O}(m)$ time. The update of the order partition (the last three lines of the loop) can be easily done in $\mathcal{O}(n)$ time. (We point out that, using more involved techniques for partition refinement [67, 65], this step could even be done in $O(|N_G(v_i)|$ time.) The choice of the vertex v_i is made by running LexBFS on the graph G_{Nxt}. Since this graph is of size $\mathcal{O}(n+m)$. The $\mathcal{O}(nm)$-time for computing σ follows.

The function **IntervalModel** constructs an interval model of $\mathtt{IG}(G, \sigma)$ based on the layout σ like in Remark 2.2.13. A single pass along $\sigma = (v_1, \ldots, v_n)$ is enough to assign to every vertex v_i the interval $[i : j]$, where j is the biggest index such that $v_i v_j \in E(G)$. This can be done in

$\mathcal{O}(deg_G(v))$-time per vertex v, where $deg_G(v)$ is the degree of v in G. In total, it gives $\mathcal{O}(n+m)$-time for computing the interval model.

□

Algorithm `MIC_Ordering`
Input: $G = (V, E)$ connected
Output: a nice layout σ and the corresponding interval model
let v_1 be the last vertex encountered by `LexBFS(`G`)`
$\rho := (v_1)$
$\mathrm{Nxt} = N_G(v_1); \mathrm{Rst} := V \setminus N_G[v_1]$
$OP := v_1 \bullet \mathrm{Nxt} \bullet \mathrm{Rst}$
for $i := 2$ **to** n **do**
  let Nxt be the class appearing after ρ in OP
  let v_i be the last vertex encountered by `LexBFS`
    launched on G_{Nxt} starting from d_2 (see Theorem 4.2.8)
  $\rho = \rho \bullet v_i$
  if $|\,\mathrm{Nxt}\,| \geq 2$ **then**
    replace Nxt in OP by $v_i \bullet (\mathrm{Nxt} \setminus \{v_i\})$
  let C be the last class of OP such that $N_G(v_i) \cap C \neq \emptyset$
  if $C \setminus N_G(v_i) \neq \emptyset$ **then**
    replace C in OP by $(C \cap N_G(v_i)) \bullet (C \setminus N_G(v_i))$
$\sigma := OP$
IntervalModel(σ)

Figure 4.3: Algorithm `MIC_Ordering`.

4.4 Conclusions

We give in this chapter an $\mathcal{O}(nm)$ time algorithm computing a minimal interval completion of an arbitrary input graph. The algorithm is based on the notion of nice layouts, which characterize a minimal interval completion, and on Theorem 4.2.8 which gives a sufficient condition for a nice layout. We point out that there are nice layouts satisfying the conditions of Theorem 4.2.8, which cannot be produced by the algorithm. Such examples can be easily obtained when the input graph is a cycle. In particular a layout produced by our algorithm is always a breadth-first search ordering, which is not required by the theorem.

There are two very natural directions for further research. One is to obtain a faster algorithm for the minimal interval completion problem. In our algorithm, each time when we choose a new vertex, we run LexBFS as the tie-break rule. A faster choice would improve the running time of the algorithm: just maintaining the ordered partition can be done in linear time [67, 65]. A naive technique would consist of doing only one sweep of LexBFS and then choosing, at each step, the vertex of Nxt with minimum LexBFS number. Unfortunately this approach does not produce a minimal interval completion.

The second important question is to characterize all nice layouts. For the minimal triangulation problem, the perfect elimination orderings (which play the same role as the nice layouts here) have been completely characterized. In our case, we have examples of nice layouts that do not satisfy the conditions of Theorem 4.2.8.

Finally, layouts in the context of interval graphs make a natural framework for tackling pathwidth. First, because pathwidth is the minimum clique number over interval embeddings of G. Then again, because an equivalent definition of pathwidth states that it is equal
$\max_{i \in 1..n} |N_G(v_{i+1}, \ldots, v_n)|$, over all layouts of G. Finally, given a prefix ρ, our algorithm finds a vertex v_i that has an inclusion minimal grey neighborhood Nxt and puts it right after ρ. This may be regarded as a naive attempt to compute the pathwidth, by trying to eliminate a vertex from $N_G(v_{i+1}, \ldots, v_n)$ as quickly as possible. It is an interesting open question if this algorithm, or maybe a slightly modified version with more sophisticated rules, would yield interval completions of small pathwidth.

Chapter 5

Pre-order - clique path encoding

Contents

We saw in Chapter 4 that clique paths, and maximal clique paths in particular, can be very useful in analysis of interval graphs. In fact, the approach to interval completion through clique paths and path decompositions was the first one, before the layouts, that gave positive results (see Section 2.3). This first polynomial time algorithm computing minimal interval completions was published in [73]. One of its main features consists in finding a clique path of an interval graph with special properties. The difficulty is that an interval graph may have a huge number of clique paths (exponential in the size of the graph). Therefore we need a compact representation of all possible clique paths. The tool we use for this purpose we call the pre-order. Apart from minimal interval completions, it seems interesting on its own and will be presented thoroughly in this chapter. The rest of the minimal interval completion algorithm is presented in Chapter 6.

There exists a classical way of representing clique paths, using PQ-trees (see [61]). This representation could be used in the algorithm of Chapter 6, as we checked after a suggestion by Christophe Paul. Nevertheless, we describe here a different approach for the clique path encoding. It is based on the notion of blocks associated to a fixed minimal separator S of G. Recall that, by Lemma 2.2.20, each separator of G appears as an edge in every clique path of G. Given a clique path P_G of G, remove all the edges that correspond to minimal separators contained in S. This partitions the clique path into subpaths. We prove that each of these subpaths corresponds to a block associated to S (see Definition 5.1.1). Let L be the list of blocks as they appear in the partition of P_G from the left endpoint up to the edge corresponding to S that we removed. Let R be an analogous list, this time taken from the right endpoint up to S. (L, R) constructed in this way is a partition of the set of blocks associated to S into two sequences, which are chains of the pre-order

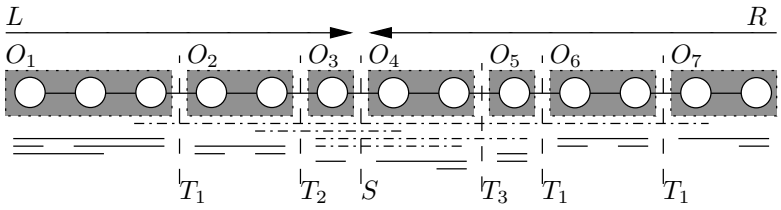


Figure 5.1: Blocks associated to a minimal separator.

that we define later (see Definition 5.1.10). Conversely, each way the blocks are partitioned into two chains, with respect to the pre-order, there is some clique path of G that respects this order. What is left to investigate, are the permutations of cliques inside each block. But this can be done recursively with the same tools. Therefore, we have control over all clique paths of G.

In Figure 5.1, we have an example of such a partition. The separators $\{T_1, T_2, T_3, S\}$ are all contained in S. The removal of the corresponding edges of the clique path P_G yields the ordered partition $(O_1, \ldots, O_7)$ of the graph G into blocks associated to S. This partition can also be seen as the pair (L, R), where $L = (O_1, O_2, O_3)$ and $R = (O_7, O_6, O_5, O_4)$. Notice that it is possible to put O_6 just to the left of O_2, keeping all the intervals contiguous; but it is not possible to put O_2 just to the right of O_4, since there is a maximal clique in O_4 which does not contain the intersection of O_2 with S. This is what our pre-order will take into account.

The reason for which we decided to keep the pre-order is twofold. First, PQ-trees are simple only if used as a black box - their construction algorithm is quite complex, whereas the pre-order processing seems quite straightforward. The other point is that the pre-order is interesting in its own right. Generalized, it gives a procedure that chooses a minimal separator S of an arbitrary graph G and computes the decomposition of G into blocks associated to S. The blocks are arranged in a path to give a path decomposition of G. Then, recursively, the blocks may again be partitioned. A similar approach using minimal separators to partition the graph was used to compute tree decompositions.

There is a greedy algorithm that tries to compute a tree decomposition of small width. It is based on efficient algorithms for finding separators of small cardinality and proceeds as follows. At each step, find a separator set S of small cardinality and compute the set of connected components $\mathcal{C}(S) = \{C_1, \ldots, C_p\}$. Add the fill edges necessary for S to be a clique. Run the procedure recursively on each subgraph $G[O_i]$ induced by a block $O_i = C_i \cup N(C_i)$ associated to S. The recursion stops when the considered graph is complete. Thanks to the fact that the separator S used at a given step is turned into a clique for the next step, we ensure that, for each $1 \leq i \leq p$, the decomposition of $G[O_i]$ obtained has a bag that contains $O_i \cap S$. This makes it possible to glue the decomposition trees of these subgraphs into a decomposition tree of the whole G. This simple principle serves as a basis for many interesting results on treewidth ([25, 24]). Despite the strong analogies between tree decompositions and path decompositions, there are no similar algorithms for computing pathwidth. We believe that our pre-order might provide a first step in this direction.

5.1 Control over clique paths

Throughout this chapter we simply use *clique path* to refer to a maximal clique path.

In this chapter we study the structure of an interval graph $G = (V, E)$ in relation with a maximal clique path P of G (see 2.2.17). For simplicity, we simple write "clique path" to denote a maximal clique path. A very important notion that we use is that of a block. The reader may refer to the work of Bouchitté and Todinca [25] for more discussion on blocks.

Definition 5.1.1. *Given an interval graph G, a block $O \subseteq V$ is the union of a minimal separator S with a full component C associated to S. S is called the* separator bordering O, *denoted $S(O)$.*

A very important feature of blocks of interval graphs is that, in every clique path P_G of G, a block O of G appears as an interval P_G^O of maximal cliques of G. P_G^O is such that $O = \mathrm{V}(P_G^O)$. Recall that $\mathrm{V}(P_G^O)$ denotes the set of vertices contained in bags of P_G^O.

Theorem 5.1.2. *Given a block O of an interval graph G, O is equal the union of the set $\mathcal{K}_O$ of maximal cliques contained in O. Moreover, these cliques appear consecutively in any clique path P of G.*

This theorem is a direct consequence of the following three technical lemmas.

Lemma 5.1.3. *Given a maximal clique K of an interval graph $G = (V, E)$, there is no full component associated to K.*

Proof. Suppose there is a full component C associated to K. Consider the graph $G' = G[K \cup C]$. Let P be some clique path of G'. K is also a maximal clique of G', so it is a node in P and there is K', another maximal clique of G', adjacent to K in P. By Lemma 2.2.20, $S = K \cap K'$ is a minimal separator of G'. So S separates some vertex $x \in K \setminus S$ from a vertex $y \in C$, which contradicts the fact that C is a full component associated to K. □

Lemma 5.1.4. *Given O, a block of an interval graph G, for any vertex $x \in O$ there is a maximal clique K of G such that $x \in K \subseteq O$.*

Proof. Suppose x is not contained in any maximal clique contained in O. If x is not contained in the separator bordering O, then any maximal clique containing x is contained in O, since x has no neighbor in $V \setminus B$. So x is contained in the separator S bordering O. Let C be the full component associated to S such that $O = S \cup C$. Consider the chordal graph $G' = G[O]$. Let Ω be a maximal clique of G' that contains S. By Lemma 5.1.3, a maximal clique cannot have an associated full component, so $\Omega \neq S$. Ω contains a vertex $c \in C$, so it also is a maximal clique of G. □

Lemma 5.1.5. *Given a block O of an interval graph G, the maximal cliques contained in O induce a subpath in any clique path of G.*

Proof. Let $O = S \cup C$, where S is the separator bordering O and C the corresponding full component. Suppose, on the contrary, that there is a clique path P in which two cliques K_1, K_2 contained in O are separated in P by a clique Ω not contained in O. By Lemma 2.2.22, it means that there are vertices v_1, v_2, with $v_1 \in K_1 \setminus \Omega$ and $v_2 \in K_2 \setminus \Omega$, such that Ω separates them in G. Consider the connected component D of $G - S$ which intersects Ω. By Lemma 2.2.23, there is $\Omega \subseteq D \cup N(D)$. So $\Omega \cap O \subseteq S$, and Ω cannot intersect every path joining v_1 to v_2 in G, since S fails to intersect every v_1, v_2-path in $G[O]$. A contradiction. □

Given a minimal separator S of an interval graph, not every component in $\mathcal{C}(S)$ is full. But for every $C \in \mathcal{C}$ with $S' = N_G(C) \subset S$, S' also is a minimal separator of G. Indeed, S' has two full components : C and D, where D is a component associated to S' which contains a full component associated to S. So $S' \cup C$ also is a block.

Definition 5.1.6. *Given a minimal separator S of an interval graph G, any block O with the bordering separator S' contained in S is a* block associated *to S. If $O \setminus S$ is a full component associated to S then O is a* full block associated *to S. The set of blocks associated to S is denoted by $\mathcal{O}(S)$.*

In this way, any minimal separator S defines a partition of the set $\mathcal{K}$ of maximal cliques of G into subsets corresponding to the blocks associated to S. Indeed, by Lemma 2.2.23, each $K \in \mathcal{K}$ is contained in exactly one element of $\mathcal{O}(S)$. Moreover, by Theorem 5.1.2, for any clique path P_G of G, the cliques contained in a block $O \in \mathcal{O}$ induce a subpath of P_G. These are the subpaths mentioned in the description opening this chapter. When we remove from P_G all the edges that correspond to separators contained in S, the set of subpaths obtained corresponds exactly to the set of blocks $\mathcal{O}(S)$.

A block O_i associated to S may be seen in two ways. First is directly the set of vertices O_i it contains. The second is the set of maximal cliques $\mathcal{K}_i$ which it contains. Notice that the union of cliques in $\mathcal{K}_i$ gives exactly O_i. So, given a clique path P_G of G, it is very easy to compute the set of blocks associated to any minimal separator S of G. Just traverse P_G from left to right. The first block starts with the first clique. It ends when we encounter an edge e_T of P_G which corresponds to a minimal separator T included in S. With the data structure presented in [9], it can be done in time linear in n.

The blocks associated to S, ordered with some clique path P_G of G give a *valid ordered partition* of the set $\mathcal{K}$ of all maximal cliques of G.

Definition 5.1.7. *Consider an ordered partition $OP = [\mathcal{K}_1, \ldots, \mathcal{K}_p]$ of $\mathcal{K}$ into blocks associated to S, such that there exists a clique path P_G of G such that the subpaths $P_G^1, \ldots, P_G^p$ formed by the maximal cliques in $\mathcal{K}_1, \ldots \mathcal{K}_p$ appear in this order (see also Lemma 5.1.5). Such an ordered partition is called* valid*.*

Observe that, when each part of a valid ordered partition OP is a singleton, OP is exactly a clique path of G. Our purpose here is to describe the way in which an ordered partition of the set of maximal cliques can be refined in order to obtain a clique path of G. Clearly for any clique path P of G, the subpath P^O of cliques contained in O defines a clique path of $G[O]$, for any block O. But the converse is not true, there are clique paths of $G[O]$ which can not be extended into clique paths of G. The next lemma characterizes the clique paths of O extendable into clique paths of the whole graph.

Lemma 5.1.8. *Let O be a block of an interval graph G.*

1. A clique path P^O of $G[O]$ can be extended into a clique path of G if and only if the separator bordering O is contained in a maximal clique at an endpoint of P^O.

2. In a clique path P_G of G, subpath P_G^O can be replaced by any clique path of $G[O]$ satisfying the above property.

Proof. Let P_G^O correspond to a subpath of P_G. Let S be the separator bordering O and C the corresponding full component. Let D be another full component associated to S and $O_D = S \cup D$.

Then P_G^O and $P_G^{O_D}$ are disjoint subpaths of P_G. If $P_G^{O_D}$ is to the right (resp. left) of P_G^O, by the properties of a clique path the leftmost (resp. rightmost) clique of P_G^O must contain $O \cap O_D = S$ and the conclusion follows.

Conversely, let P^O be any clique path of $G[O]$ such that S is contained at one endpoint. For any clique path $P_G^{'}$ of G, we can replace the subpath $P_G^{'O}$ formed by the cliques of O with P^O without violating the properties of a clique path. Indeed, the vertices of G that belong to some cliques at both sides of $P_G^{'O}$ in $P_G^{'}$ are contained in every maximal clique of O. So putting the endpoint of P^O that contains S next to a clique (not contained in O) that also contains S ensures that the properties of clique path are satisfied. □

This lemma gives us conditions according to which we can refine a valid ordered partition. Let us now focus on obtaining the first refinement of $\mathcal{K}$ as some order on $\mathcal{O}(S)$.

From now on we fix a separator S. Consider a valid ordered partition $OP = [\mathcal{K}_1, \ldots, \mathcal{K}_p]$ based on $\mathcal{O}(S)$, that is to say that $\mathcal{K}$ is partitioned into sets of maximal cliques that correspond to the blocks associated to S. Moreover, assume that the order on $\mathcal{K}_i$ reflects the order in which the corresponding blocks appear in a clique path P of G. In such a situation we say that the order of blocks in OP is defined by P.

Instead of working with the ordered partition OP, we work with the pair (L, R) where L is the list of blocks of $\mathcal{O}(S)$ situated to the left of S in OP, ordered from left to right; R is the list of blocks to the right of S, from right to left. Speaking of "to the left of S" or "to the right of S" refers to the fact that the cut is made between two blocks the intersection of which equals S (see Figure 5.1). Notice that, whereas OP is a valid ordered partition of the set of maximal cliques of G into sets of cliques contained in different blocks associated to S, the corresponding pair (L, R) is directly an ordering of blocks associated to S. Actually, for a valid ordered partition OP, the corresponding pair (L, R) defines a path decomposition of G. Indeed, it is easy to verify that $L \bullet \overleftarrow{R}$, the concatenation of L with the reverse ordering of R, is a path decomposition of G. The edge between L and $\overleftarrow{R}$ represents S. The two approaches are equivalent due to Theorem 5.1.2. More formally,

Definition 5.1.9. *Let $OP = (\mathcal{K}_1, \ldots, \mathcal{K}_p)$ be a valid ordered partition based on $\mathcal{O}(S)$. Let $\mathcal{K}_i, \mathcal{K}_{i+1}$ be two parts consecutive in OP such that $\mathcal{K}_i \cap \mathcal{K}_{i+1} = S$. Take $L = [O_1, O_2, \ldots, O_i]$ and $R = [O_p, O_{p-1}, \ldots, O_{i+1}]$, where O_j is the block associated to S corresponding to the part $\mathcal{K}_j$. (L, R) is a* valid pair *based on S.*

Notice that it is possible to "break" OP at any i such that $\mathcal{K}_i \cap \mathcal{K}_{i+1} = S$. So a valid pair (L, R) is not unique for a given OP based on S. For the ease of description, let us use the same notation for a blocks O_i and the corresponding set of maximal cliques of G contained in O_i. Thus, hereafter O_i replaces $\mathcal{K}_i$.

Clearly the pair (L, R) identifies the ordered partition OP. Now given two lists L and R such that each element of $\mathcal{O}(S)$ is in exactly one list, we want to know if the pair (L, R) defines a valid ordered partition. Consider two blocks O_i and O_j in $\mathcal{O}(S)$. We want to know whether there exists a clique path P_G of G in which the subpath P^{O_j} is between P^{O_i} and some edge e_S corresponding to S (the intersection of cliques incident to e_S equals S). The answer is yes only if $O_i \preceq O_j$ according to the pre-order defined as follows:

Definition 5.1.10. *Two blocks $O_i, O_j \in \mathcal{O}(S)$ satisfy $O_i \preceq O_j$ if every vertex of $S \cap O_i$ is universal in O_j (compare Lemma 2.2.19).*

Now we can state the main theorem of this chapter. It claims that with the pre-order we can capture all possible clique paths of G.

Theorem 5.1.11. *Let L, R be two lists such that each block of $\mathcal{O}(S)$ appears in exactly one of them. Then (L, R) is a valid pair based on S if and only if L and R form a partition of the partially pre-ordered set $(\mathcal{O}(S), \preceq)$ into two chains.*

Proof. For the "only if" part, note that for two blocks O_i and O_j there exists a clique path in which P^{O_j} is between P^{O_i} and some edge e_S corresponding to S only if $O_i \cap S$ is contained in every maximal clique of $G[O_i]$. Hence L and R must be chains of $(\mathcal{O}(S), \preceq)$.

Conversely, if we partition $(\mathcal{O}(S), \preceq)$ into two chains (L, R) we can construct a clique path of G in which the subpaths corresponding to blocks in L (resp. R) are situated to the left (resp. right) of e_S, joined by this edge, respecting the order of the chains L, R. In this setting, we only need to get a clique path for each block $O \in \mathcal{O}(S)$ that lets us glue them all together into a clique path of G.

By using Lemma 5.1.8, we can construct a clique path of every $O \in \mathcal{O}(S)$ with $S(O)$ at one end. The conclusion follows. □

The first level of refinement is completed. We have an ordered partition of the set $\mathcal{K}$ of all maximal cliques of G, obtained as a partition of $(\mathcal{O}(S), \preceq)$ into two chains. Now we need to recursively refine each of the blocks $O \in \mathcal{O}(S)$ until we reach a partition into single maximal cliques, thus a clique path of G. We need to verify that each clique path $P_{G[O]}$ of $G[O]$, for $O \in \mathcal{O}(S)$ that we obtain can be used as a subsequence in the clique path of G under construction. For this purpose we can use a bijection between the set of clique paths of $G[O]$ satisfying this condition and the set of all clique paths of the auxiliary graph $G^+[O]$ defined as follows:

Definition 5.1.12. *Given a minimal separator S of an interval graph G and a full block O associated to S, the graph $G^+[O]$ is the graph $G[O]$ augmented with a dummy vertex d adjacent to every vertex in S.*

With this definition, the following lemma gives the above mentioned bijection. Thanks to this correspondence, we can analyze the clique paths of $G[O]$ extendable to a clique path of G with the same tools that we use for unconstrained clique paths.

Lemma 5.1.13. *Let O be a block associated to S, a minimal separator of an interval graph G. P is a clique path of $G[O]$ with the leftmost bag containing $S' = O \cap S$ iff $(\{d\} \cup S') --P$ is a clique path of $G^+[O]$. Moreover, every clique path of $G^+[O]$ is of the form $(\{d\} \cup S') --P$, where P is a clique path of G, modulo taking the reverse.*

Proof. The first part of the claim is straightforward from the properties of clique path. Then, since O is a full block associated to S, S is not a separator in $G[O]$. So, by construction, the maximal clique $K = S \cup \{d\}$ is not a separator in $G^+[O]$. Thus, by Lemma 2.2.22, K has to be an endpoint in every clique path of $G^+[O]$. □

Now, using Theorem 5.1.11 and Lemmas 5.1.8 and 5.1.13 it is not difficult to verify the theorem that summarizes the argumentation given in this section:

Theorem 5.1.14. *Any clique path of G can be obtained by the following algorithm. First, a minimal separator S of G is chosen and an ordered partition OP of $\mathcal{K}$ is computed, according*

to Theorem 5.1.11, by fixing a partition of $(\mathcal{O}(S), \preceq)$ into two chains. Then, the same procedure is recursively applied to the graph $G^+[O]$, for each block O_i corresponding to a part O_i in OP, according to Lemmas 5.1.8 and 5.1.13, to obtain a refinement of O_i into a clique path of $G[O]$.

5.2 Structure of the pre-order

5.2.1 The co-comparability graph

In this subsection, we present a series of observations on the pre-order that we will use in Chapter 6 to prove correctness of the algorithm computing minimal interval completions. In order to manipulate the pre-order, it is useful to analyze deeper the structure of this relation. Let us define the *incomparability* relation $\|$ such that $O_1 \| O_2$ if and only if $O_1 \not\preceq O_2$ and $O_2 \not\preceq O_1$. Let $(\mathcal{O}(S), \|)$ be the graph on the vertex set $\mathcal{O}(S)$ in which two vertices are adjacent if and only if they are incomparable. This graph is the co-comparability graph of $(\mathcal{O}(S), \preceq)$. Note that $O_1 \| O_2$ means that O_1, O_2 must be at different sides of S in valid pair (L, R). In particular, there cannot be three pairwise incomparable blocks.

The pre-order is a partial pre-order of width 2 (maximum cardinality of an anti-chain in $\preceq$ equals 2) and, as such, can be decomposed into a sequence of disjoint components that are totally pre-ordered in a natural way, based on $\preceq$. This structure permits to efficiently analyze all possible partitions of $(\mathcal{O}(S), \preceq)$ into two chains.

The above mentioned components are the connected components of the co-comparability graph of $(\mathcal{O}(S), \preceq)$.

The following lemma shows that $\preceq$ is a pre-order of width 2. Thus, by Dilworth's theorem [37], $\preceq$ can always be partitioned into two chains. Moreover, it describes the way the graph $(\mathcal{O}(S), \|)$ reflects the possible positions of blocks, in one of the chains in (L, R).

Lemma 5.2.1. *Let S be a minimal separator of an interval graph G. The graph $(\mathcal{O}(S), \|)$ is bipartite. If two blocks $O_1, O_2 \in \mathcal{O}(S)$ are forced to be in the same color class (equivalently, they are joined in $(\mathcal{O}(S), \|)$ by a path of even length) then they appear in the same chain in any partition of $\mathcal{O}(S)$ into (L, R). If O_1, O_2 are forced to be in different color classes (equivalently, they are joined in $(\mathcal{O}(S), \|)$ by a path of odd length) then they appear in different chains in any partition of $\mathcal{O}(S)$ into (L, R).*

Proof. According to Lemma 5.3.2, if two blocks O and O' are incomparable with respect to the $\preceq$ relation, they cannot appear on the same side of any edge e_S corresponding to S in any clique path P_G of G. Thus, if we fix an edge e_S representing S, by assigning color "L" to all blocks to the left of e_S in P_G, and color "R" to all those to its right, we obtain a two-coloring of $(\mathcal{O}(S), \|)$.

The next observations come directly from the fact that if O and O' must be in the same color class (resp. in different color classes), then there is a path of even (resp. odd) length from O to O' in $(\mathcal{O}(S), \|)$. We then use the fact that two incomparable blocks must be on different sides of e_s. □

Notice that it is inside a connected component of $(\mathcal{O}(S), \|)$ that some forcing between two blocks O_i, O_j, regarding they relative positions in (L, R) may appear. So if O_i, O_j belong to different components of $(\mathcal{O}(S), \|)$, then there are both partitions where O_i, O_j belong to different chains and such where they appear in the same chain. The following lemma adds to this autonomy the fact that the components of $(\mathcal{O}(S), \|)$ cannot be interlaced in $\preceq$.

Lemma 5.2.2. *Let C and D be two distinct connected components of $(\mathcal{O}(S), \|)$. There is no triple of one-blocks O_c, O'_c, O_d such that $O_c, O'_c \in C$, $O_d \in D$ and $O_c \preceq O_d \preceq O'_c$.*

Proof. Suppose that such a triple exists and consider a path P joining O_c and O'_c in $(\mathcal{O}(S), \|)$: $P = [O_c = O_1, O_2, \ldots, O_p = O'_c]$. Then there are two consecutive one-blocks O_i and O_{i+1} on this path such that $O_i \preceq O_d \preceq O_{i+1}$. By the transitivity of the $\preceq$ relation, $O_i \preceq O_{i+1}$, contradicting the fact that the two one-blocks are adjacent in $(\mathcal{O}(S), \|)$. □

So the pre-order can be extended in a natural way to the set of components of $(\mathcal{O}(S), \|)$.

Definition 5.2.3. *Define the order $(\mathcal{C}(S), \leq)$ with $\mathcal{C}$ being the set of connected components of $(\mathcal{O}(S), \|)$ and $C \preceq D$, $C, D \in \mathcal{C}$ if $O_c \preceq O_d$ for all $(O_c, O_d) \in (C, D)$.*

Indeed, by Lemma 5.2.2 we deduce directly :

Lemma 5.2.4. *For two distinct connected components C and D of the graph $(\mathcal{O}(S), \|)$, we have $C \preceq D$ or $D \preceq C$. Moreover, if there are two blocks $O_c \in C$ and $O_d \in D$ that are equivalent with respect to $\preceq$, then each of the components is composed of a single block.*

The following lemma states that blocks which are equivalent with respect to $\preceq$ behave the same with respect to connected components of $(\mathcal{O}(S), \|)$.

Lemma 5.2.5. *Let $\mathcal{O}_{eq}$ be a set of blocks equivalent for $\preceq$. Then they are either in separate singleton connected components of $(\mathcal{O}(S), \|)$ or they are forced to be in the same color class of one connected component of $(\mathcal{O}(S), \|)$.*

Proof. Let $O_{eq} \in \mathcal{O}_{eq}$. If there is $O \| O_{eq}$ in $\mathcal{O}(S)$ then O is incomparable with the rest of one-blocks in $\mathcal{O}_{eq}$. Thus all $\mathcal{O}_{eq}$ are in the same connected component of $(\mathcal{O}(S), \|)$, in the opposite color class than O. Else, there is no $O \| O_{eq}$. Hence $\mathcal{O}_{eq}$ form singleton connected components of $(\mathcal{O}(S), \|)$. □

In this document we extend the notion of a topological sorting to graphs of partial pre-orders.

Definition 5.2.6. *Given a graph G of a partial pre-order $\preceq$, σ is a topological sorting of G if for every pair of elements which are not equivalent with respect to $\preceq$, $O_i \preceq O_j$ implies that O_i appears before O_j in σ.*

Notice that for blocks that are equivalent with respect to $\preceq$, they relative position in a topological sorting of $(\mathcal{O}(S), \preceq)$ is not specified.

It is a direct consequence of Lemma 5.2.4 that the elements of a connected component of $(\mathcal{O}(S), \|)$ are consecutive in any topological sorting of $(\mathcal{O}(S), \preceq)$.

Lemma 5.2.7. *The set of blocks in a connected component of $(\mathcal{O}(S), \|)$ appears consecutively in any topological sorting of $(\mathcal{O}(S), \preceq)$. Moreover, a topological sorting of $\preceq$ induces a linear order $C_1, \ldots, C_p$ on the connected components of $(\mathcal{O}(S), \|)$.*

This completes the structural description of the pre-order.

`procedure` **bipartition**
`Input:` $\mathcal{O}$ one-blocks sorted in $\trianglelefteq$
`Output:` L, R - two lists of one-blocks, partition of $\mathcal{O}$ into two color classes
`Variables:`
$fixL, fixR$ - lists of one-blocks fixed to be in the corresponding color classes; the top elements of $fixL, fixR$ form the topmost couple of incomparable elements
temp - lists of one-blocks after both elements of MA in $\trianglelefteq$, temporarily put in L
$sL = (oL, cL), sR = (oR, cR)$ - for every one-block, we store pointers to the one-blocks just below it in (L, R) together with the information if they are smaller or incomparable. For example, $sL(O_i) = (O_j, true)$ means that the maximum one-block in L, smaller than O in $\trianglelefteq$, is O_j and $O_j \preceq O_i$

$fixL \leftarrow \emptyset; fixR \leftarrow \emptyset$
$temp \leftarrow \emptyset$
`procedure` **updadeLR**$(temp)$
$first = head(temp)$
$(sL, sR)(first) = (sR, sL)(first)$
`forall` $O \in \mathcal{O}$ `do`
    $(sL, sR)(O) = (sL(first), sL(O))$
`end`
`forall` $O \in \mathcal{O}$ `do`
    compute $sL(O)$
    compute $sR(O)$
    `if` $(cL(O))$ `then` //case 1
        temp = concat(temp,O)
        `if` (not cR(O)) `then`
            $fixL = concat(fixL, temp)$
            $temp = \emptyset$
        `endif`
    `else if` $(oR(O))$ `then` //case 2
        $fixR = concat(fixR, O)$
        $fixL = concat(fixL, temp)$
        $temp = \emptyset$
    `else` //case 3
        $updateLR(temp)$
        $fixR = concat(fixR, temp)$ $fixL = concat(fixL, O)$
        $temp = \emptyset$
    `endif`
`endforall`
$L = concat(fixL, temp)$
$R = fixR$

Table 5.1: Bipartition algorithm.

5.3 Learning the pre-order

It is costly to learn the pre-order $(\mathcal{O}(S), \preceq)$ in a naive way. There can be $\Omega(n)$ blocks. Their intersections with S can also be of cardinality $\Omega(n)$. $\Omega(n^2)$ comparisons of linear cost give $\Omega(n^3)$ time complexity. In this section, we first show a way to obtain a topological sorting of $(\mathcal{O}(S), \preceq)$ in linear time. Then we describe how to use it to learn the pre-order using linear number of comparisons. Finally, we give a sketch of the proof that the pre-order can be computed in linear time.

The reader may skip this section and continue in Chapter 6 with the results using the pre-order to compute a minimal interval completion. We give here some technical analysis and present a linear time algorithm that computes the pre-order $(\mathcal{O}(S), \preceq)$ for a minimal separator S of a chordal graph G.

The algorithm we present produces a list of connected components of $(\mathcal{O}(S), \|)$, in a topological sorting (see Lemma 5.2.7). Each component C is represented by two lists of blocks, also arranged in a topological order, that correspond to a bipartition of C in $(\mathcal{O}(S), \|)$. The first step is to obtain a topological sorting of $(\mathcal{O}(S), \|)$. Then we use it to compute the equivalence classes, according to $\preceq$. Blocks grouped in equivalence classes form, in a natural way, a partial order.

For the sake of simplicity, let us use the same notation for the relation on equivalence classes as for the relation on single blocks. Notice that in case of singleton equivalence classes they correspond to the same thing. Regarding the equivalence classes as single elements transforms the pre-order from a partial pre-order to a partial order, which makes the use of standard poset terminology more relevant. Therefore we allow this little abuse of terminology, and signalize that it does not affect the correctness of presented claims.

The second step is to bipartition the list of blocks, thus to produce two lists corresponding to two color classes of $(\mathcal{O}(S), \|)$. The third step is to compute the connected components. To achieve this, we traverse the two lists (color classes) and find where the connected components start and finish. At the end, each equivalence class may be replaced with any permutation of its elements.

5.3.1 Topological order

Let us define two notions that are useful for the computation of the pre-order.

Definition 5.3.1. *For every $O \in \mathcal{O}(S)$ let*

$$big(O) = \max\{K \cap S | K \in \mathcal{K}, K \subseteq O\}, \quad small(O) = \min\{K \cap S | K \in \mathcal{K}, K \subseteq O\} \tag{5.1}$$

The maximum and minimum are taken with respect to set inclusion. Notice that they are well defined, since for the set of cliques contained in a block O associated to S, the intersections of cliques with S are totally ordered by inclusion. Consider a clique path P of G with a fixed edge e_S corresponding to S (the intersection of incident bags equals S). Pick any block O associated to S. Recall that, by Theorem 5.1.2, the maximal cliques contained in O are consecutive on P. S is not a u, v-separator for any $u, v \in O$, since O is a block associated to S. So, by Lemma 2.2.21, all bags corresponding to cliques in O have to appear as a subpath P_G^O at one side of e_S in P_G. W.l.o.g. assume it is the left hand side. In this setting, $big(O)$ corresponds to the edge e_b of P incident to the rightmost clique of P_G^O, to the right. If there are other bags to the left of P_G^O in P_G, then $small(O)$ corresponds to the edge e_s of P_G incident to the leftmost clique of P_G^O, to the left. Hence, by Lemma 2.2.20, $small(O)$ also is a minimal separator of G. Only if O is one of the two blocks that contain the endpoints of P_G, $small(O)$ may not be a minimal separator of G.

It is worth mentioning here, that clique path is a very useful and efficient tool for computations on interval graphs (like clique tree for chordal graphs). Especially, with the $\mathcal{O}(n)$ space representation given by Berry et al. in [9]. It is not difficult to verify that, with this tool, the set of blocks $\mathcal{O}(S)$ associated to a minimal separator S, and the corresponding values of $big(O)$ and $small(O)$, for $O \in \mathcal{O}(S)$ can be computed in time $\mathcal{O}(n)$. This is a good starting point for working on the pre-order, since the following characterization holds.

Lemma 5.3.2. *Let O_i and O_j be two blocks in $\mathcal{O}(S)$. Then $O_i \preceq O_j$ iff $big(O_i) \subseteq small(O_j)$.*

Proof. Assume that $O_i \preceq O_j$. By Definition 5.1.10, $S(O_i)$ is contained in every maximal clique contained in O_j, thus $big(O_i) \subseteq small(O_j)$.

Conversely, assume that $big(O_i) \subseteq small(O_j)$. So $S(O) = big(O)$ is contained in the minimum over the intersections of maximal cliques of G contained in O_j with S, hence $S(O)$ is contained in each clique in O_j. □

Let us define a relation on the set of blocks that we use to obtain a topological sorting of $(\mathcal{O}(S), \preceq)$.

Definition 5.3.3. *Two blocks satisfy $O_i \trianglelefteq O_j$ if $\triangle(O_i) \leq \triangle(O_j)$, where $\triangle(O_i) = |big(O_i)| + |small(O_i)|$.*

It is an easy observation, that the $\trianglelefteq$ relation is a linear pre-order on the set of blocks. Moreover:

Lemma 5.3.4. *The relation $\trianglelefteq$ with each equivalence class permuted in an arbitrary way gives a topological sorting of $\preceq$.*

Proof. Let $O_1, O_2 \in \mathcal{O}(S)$. $O_1 \preceq O_2$ means that $big(O_1) \subseteq small(O_2)$. Therefore

$$|small(O_1)| \leq |big(O_1)| \leq |small(O_2)| \leq |big(O_2)|$$

$$|small(O_1)| + |big(O_1)| \leq |small(O_2)| + |big(O_2)|$$

so $O_1 \trianglelefteq O_2$. □

Notice that $\triangle(O)$ is bounded by $2n$. Therefore, the elements of $\mathcal{O}(S)$ can be sorted, according to $\trianglelefteq$, by an algorithm like bucket-sort, in time linear in n. Computing the values of $\triangle(\cdot)$ can be done in time linear in m.

Let us finish the discussion on the topological sorting with the following lemma. It clarifies the structure of equivalence classes with respect to $\trianglelefteq$ in comparison with equivalence classes with respect to $\preceq$.

Lemma 5.3.5. *An equivalence class $\mathcal{O}_{eq}^{\trianglelefteq}$, with respect to $\trianglelefteq$, consists of at most two whole equivalence classes of the pre-order $\preceq$. Moreover, if there are two of them, then they are incomparable with respect to $\preceq$.*

Proof. Suppose $\mathbf{O}$ is such an equivalence class. $O_1, O_2 \in \mathbf{O}$ means that

$$|big(O_1)| + |small(O_1)| = |big(O_2)| + |small(O_2)| \tag{5.2}$$

If two elements are equivalent for $\preceq$ then, clearly, they also are equivalent for $\trianglelefteq$. So the equivalence classes of $\trianglelefteq$ form a partition of equivalence classes for $\preceq$. On the other hand, $\mathbf{O}$ can contain two equivalence classes of $\preceq$ only if the elements are incomparable between them. Indeed, the strict inclusion $big(O_1) \subsetneq small(O_2)$ is in contradiction with Equation 5.2. Finally, since $\preceq$ is a partial pre-order of with 2, there may be at most two mutually incomparable elements. □

The representant of **O** that comes first in the topological sorting can be used in $\preceq$-comparisons to filter the elements of **O** into the same or the second equivalence class. This can be done in $\mathcal{O}(n)$ number of $\preceq$ comparisons.

5.3.2 Bipartition

To construct the two lists of blocks that correspond to a bipartition of $\mathcal{O}(S)$ we take the list of blocks yielded by the previous step. The construction of the left and right chain is done incrementally. Control over the structure is exercised with the topmost maximum anti-chain among the elements considered so far. Since $\preceq$ is of width two, throughout the algorithm we keep track of the position of topmost couple of incomparable blocks TA.

Each time a new element O is considered, we check its relation with the top elements of the lists constructed so far. If O is $\preceq$-bigger than the top left or top right element, we just put it over it. If O is not comparable with the top element on the other side, then we update the topmost anti-chain TA. If O is incomparable with both top elements, then we use TA to rearrange the partition. The sub-chains constructed up to the intersection with TA (inclusive) are denoted $fixL$ and $fixR$. The chain of elements $\trianglelefteq$-bigger than both elements of TA considered so far is denoted *temp* and is considered to be above $fixL$.

The following lemma states that the elements $\trianglelefteq$-bigger than TA, in a partition of elements $\trianglelefteq$ smaller than O into two chains, can always be rearranged in a way which lets append O at the top of one of the chains.

Lemma 5.3.6. *Let O be an element of the considered partial order $(\mathcal{O}(S), \preceq)$ of width 2. Let $fixL$ denote the sub-chain of the left chain up to TA (including the intersection with TA) and $fixR$ the analogous chain on the right side. Then the bipartition can be extended by appending O to one of $fixL, fixR$ and the chain temp of the other elements considered so far to the other.*

Proof. Let *temp* denote the list of the elements strictly above TA considered so far, $\preceq$-ordered. They form a chain, since TA is the topmost anti-chain of width 2. Its bottom element *bottom*(*temp*) is comparable with both elements of TA. O is comparable with at least one element of TA (width 2), so it can be placed above it. We append *temp* above the other and the construction is finished. □

Now we can prove the following theorem.

Theorem 5.3.7. *The algorithm bipartition produces a proper 2-coloring of $\mathcal{O}$.*

Proof. Clearly, the set of conditions present in the algorithm is complete, i.e. it covers all the possible situations when a new block O is considered. In first two cases, O is comparable with a maximal element already placed on the left or right side. So O can be put just above it and we maintain a partition of $\mathcal{O}$ already considered into two chains. In the third case, O is incomparable with both $oL(O)$ and $oR(R)$. On the other hand, throughout the execution of the algorithm, *top*($fixL$) and *top*($fixR$) form the topmost anti-chain among the elements processed so far (if both are not null). Indeed, it is an easy observation that each time $fixR$ or $fixR$ is modified, an anti-chain is put at the top and TA is updated. And it happens each time a new anti-chain appears. The algorithm rearranges the partition as described in Lemma 5.3.6.

After processing all blocks the algorithm yields a partition as needed. □

5.3.3 Components of $(\mathcal{O}(S), \|)$

Having a decomposition of $\mathcal{O}(S)$ into two chains, we can compute the list of connected components of the incomparability graph. We process the two lists of blocks, given as input. For each list we maintain a pointer. With the pair of pointers, we scan $\mathcal{O}(S)$ in a balanced way. That is to say that at each step we take a new element O_i in the $\trianglelefteq$-sorted $\mathcal{O}(S)$ and take O_L (O_R) to be the top one-block in L (R) before O_i in the sorting. That is $O_L = oL(O_i)$, $O_R = oR(O_i)$. The following lemma gives a condition that characterizes where two different connected components of $(\mathcal{O}(S), \|)$ meet.

Lemma 5.3.8. *Let (L, R) be a decomposition of a partial order $(\mathcal{O}(S), \preceq)$ into 2 chains. Let $(downL, upL)$, $(downR, upR)$ denote a bisection of the chains by some scan-line. Then there is no connected component of $(\mathcal{O}(S), \|)$ crossing this scan-line iff the bottom elements above the scan-line are all comparable with top elements below the scan-line.*

Proof. Suppose there is a component C crossing the scan-line SL. Then there are some elements $O_1, O_2 \in C$ such that the scan-line separates them. O_1, O_2 are joined by a chain of incomparabilities that crosses SL at some point. Let O_1' be the element of this chain just below SL and O_2' be the one just above, joined by incomparability. On the other hand, the elements just below the scan-line are comparable to the ones just above. Therefore, every element below the scan-line is comparable with all elements above it. A contradiction with the incomparability crossing the scan-line.

Conversely, if there is an element O_1 just above the scan-line incomparable with O_2 just below it, then they both belong to the same connected component of $(\mathcal{O}(S), \|)$. A contradiction with the fact that there is no component crossing the scan-line. □

Therefore, if O_L and O_R are comparable with (smaller than) both blocks O_L^+, O_R^+ just above, with respect to $\preceq$, then the current connected component is closed (composed of the one-blocks between the previous component and the current scan-line).

It is only for O_R^+ that we need to compute the comparison with O_L, since the other three were computed during the creation of L, R. In total, for every element we do at most three comparisons with the elements smaller in the topological sorting.

5.3.4 Complexity

We have shown that it is possible to learn the pre-order with $\mathcal{O}(n)$ time preprocessing and linear number of comparisons of type $big(O_i) \subseteq small(O_j)$. Moreover, we have said that $big(O_i)$ and $small(O_i)$ are minimal separators of G, with the only exception for $small(O_j)$, where O_j is one of two blocks associated to $\mathcal{O}(S)$ containing the endpoints of P, for any fixed clique path P of G. By a result of Ibarra [76], such inclusion can be tested in constant time.

Lemma 5.3.9. *[76] Let G be an interval graph. There is a linear time preprocessing that permits to test in constant time if $S \subseteq T$ for any minimal separators S, T of G.*

By Algorithm 5.1 every block is compared a constant number of times with the other blocks. So without further analysis, we can afford $\mathcal{O}(n)$ inclusion tests for comparing the external blocks in the given clique path P of G and state that the pre-order can be computed in linear time.

Theorem 5.3.10. *Given an interval graph G and a minimal separator S of G, the pre-order $(\mathcal{O}(S), \preceq)$ can be computed in $\mathcal{O}(n + m)$ time.*

Proof. A linear size data structure to represent a clique path of G can be created in linear time (see [9]). Then the blocks associated to S can be computed in $\mathcal{O}(n)$ time. After linear time preprocessing, the pre-order can be computed with $\mathcal{O}(n)$ number of constant time comparisons and constant number of $\mathcal{O}(n)$ time comparisons. The conclusion follows. □

Chapter 6

Minimal Interval Completion Through Decomposition

Contents

In this Chapter we present the first, to our knowledge, published algorithm computing a minimal interval completion of an arbitrary graph [73]. Despite the strong resemblance between interval completions and chordal completions (triangulations), the work on minimal triangulations, which started in the seventies of the last century with the results of Rose, Tarjan and Leuker [107], was not extended to interval completions until the year 2005. One reason for this might be that minimal triangulations have a characterization, which is often used in analysis of chordal completion, and the analogue for interval completion does not hold. It says that a chordal super-graph H of G is a minimal triangulation of G if and only if there is no edge $e \in E(H) \setminus E(G)$ such that $H - e$ is also a chordal graph. This makes it very simple to verify if a given triangulation is minimal. On the other hand, it happens that an interval super-graph H of G is such that, for any edge $e \in E(H) \setminus E(G)$, $H - e$ is not an interval graph, but still H is not a minimal interval completion of G. We also tried other approaches inspired by the similarity of interval and chordal graphs. For example, we wanted to use the characterization of interval graphs saying that these are exactly chordal, AT-free graphs (see Theorem 2.2.25). But the attempts to first triangulate, then remove asteroidal triples by adding some edges or vice-versa failed. Even more sophisticated algorithms, trying to do both at the same time, were of no use. Eventually, we tried the approach presented in this chapter.

Our algorithm proceeds in an incremental manner. It first fixes an arbitrary ordering $(v_1, \ldots, v_n)$ and, at each step i, it computes a minimal interval completion H_i of the graph $G[\{v_1, \ldots, v_i\}]$ induced in G by the set of vertices considered so far. When the vertex v_{i+1} is considered, the computation is not made from scratch, but the completion H_i is used. Therefore, a single step consists in finding a minimal interval completion of an interval graph H_i augmented with the vertex v_{i+1} adjacent to its neighborhood in G restricted to $\{v_1, \ldots, v_i\}$ (so this set is equal to $N_G(v_{i+1}) \cap \{v_1, \ldots, v_i\}$). The solution to this problem is based on the clique paths of H_i browsed with the help of the pre-order, described in the previous chapter.

6.1 Incremental approach

Notation 1. *Let us fix an ordering $(v_1, \ldots, v_n)$ of $V(G)$. We use G_i denote the subgraph of G induced by $\{v_1, \ldots, v_i\}$. Moreover, we use H_i to denote the interval completion of G_i computed by the algorithm.*

Let us first justify that the incremental approach is possible. The next lemma states that a minimal interval completion H_{i+1} of G_{i+1} can be computed from the minimal interval completion H_i of G_i computed at the previous step, by only adding fill edges incident to v_{i+1}.

Lemma 6.1.1. *Let H be a minimal interval completion of an arbitrary graph G. Let G' be a graph obtained from G by adding a new vertex x, with neighborhood $N_{G'}(x)$. There is a minimal interval completion H' of G' such that $H' - x = H$.*

Proof. Let H'' be the graph obtained by adding x and the edges between x and every element in $N_{G'}(x)$ to H. Observe that an interval completion of H'' can be obtained by only adding edges incident to x. For example, we can make x adjacent to every vertex of H''. Clearly the result, denoted by $H * x$, is an interval graph, since, given an interval model of H, it is easy to obtain an interval model of $H * x$. For example, we can add a new interval corresponding to x that starts before and ends after any interval of the model of H. If the resulting graph is not a minimal interval completion of H'', then it properly contains one. Let H' denote this minimal interval completion, with $H' \subseteq H * x$. By construction, all the edges in $E(H') \setminus E(H)$ are incident to x.

Let H' be a minimal interval completion of H'' obtained by only adding edges incident to x. We claim that H' is a minimal interval completion of G'. Suppose it is not. So there is H''', a minimal interval completion of G', with $H''' \subset H'$. If all edges in $E(H') \setminus E(H''')$ are incident to x, then H''' is also an interval completion of H'' - a contradiction with the minimality of H'. So there is an edge $e \in E(H') \setminus E(H''')$ which is not incident to x, hence e is already present in H. Let $\overline{H} = H''' - x$. It is an interval graph, as an induced subgraph of an interval graph. By construction, there is $G \subseteq \overline{H}$. Moreover, $\overline{H} \subset H$, since $e \in H \setminus \overline{H}$. This contradicts the minimality of H. □

Hence, for computing a minimal interval completion of G, we introduce the vertices of G one by one in the order $(x_1, x_2, \ldots, x_n)$. Given a minimal interval completion H_i of G_i, we compute an interval completion H_{i+1} of G_{i+1} by adding vertex x_{i+1} and the edges between x_{i+1} and $N_{G_{i+1}}$ to H_i together with a well chosen set of additional edges incident to x_{i+1}.

From now on we consider as input an interval graph $G = (V, E)$. A new vertex x is added to G, together with a set of edges incident to x. For the rest of this chapter, let G' denote this graph. We want to compute a minimal interval completion H of G', obtained by only adding edges incident to x. We say that such a minimal interval completion *respects* G.

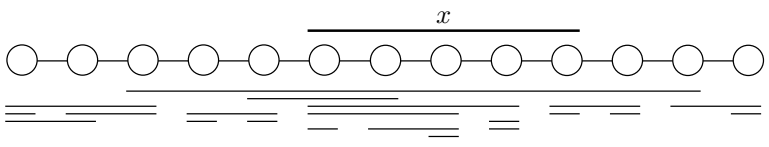


Figure 6.1: Pruning x from a clique path of G.

6.2 Principles of the algorithm

Before getting into details, let us give the general idea of the algorithm. Let G' be a graph such that $G = G' - x$ is an interval graph, for a fixed $x \in V$. Let H be a minimal interval completion of G'.

Take any clique path P_H of H. By property of clique paths, the cliques containing x form a subpath P_H^x of P_H. Now, let us get back to G. Delete x from every bag in P_H, and possibly remove the bags that do not correspond to maximal cliques of G. This yields P_G - a clique path of G, which gives a linear ordering of maximal cliques of G. We say in this case that P_G was obtained by *pruning* the vertex x from P_H.

In Figure 6.1, there is a clique path of H. The corresponding clique path of G is obtained by removing x from all bags (just remove the interval corresponding to x from the interval model).

Definition 6.2.1. *An interval completion H of G'* respects *a clique path P_G of G if P_G can be obtained by pruning x from some clique path of H.*

Clearly the maximal cliques that come from P_H^x still form a subpath of P_G. Our aim is to do the converse: to find a clique path P_G of G and a subpath P_G^x of P_G in which, by adding vertex x to every bag and possibly creating new bags at the ends of P_G^x, we obtain a clique path of a minimal interval completion of G' respecting P_G. Intuitively, P_G has to be such that P_G^x is inclusion minimal over all clique paths of G.

In this section we focus on the description of the two clique paths P_H, P_G. Given a clique path P_G of G, such that an interval completion H of G respects P_G, we describe how to reconstruct a clique path P_H. In particular, provided that we have a "nice" clique path P_G of G, this construction shows how to compute an interval model of a minimal interval completion of G.

Let us formalize the notions used in our discussions. In particular, we introduce the maximal cliques K_L, K_R of G that delimit the interval of bags which will have the vertex x added in the process of reconstruction of H.

Definition 6.2.2. *A clique path P_G is a* nice clique path *of G if there exists a minimal interval completion H respecting P_G. Let K_L (resp. K_R) of P_G be the leftmost (resp. rightmost) bag K in P_G such that x has a neighbor in K which does not belong to any bag to the right (resp. left) of K.*

In Figure 6.2, there are two clique paths of the same graph G. The neighbors of x are marked with thick lines. Notice that the set of maximal cliques inside the $[K_L : K_R]$ interval of the top clique path is strictly included in the $[K_L : K_R]$ interval of the bottom clique path.

A natural question that comes here is if K_L, K_R are well defined. Clearly, they always exist. The only difficulty comes from the case where K_L is to the right of K_R in P_G, thus the interval $[K_L : K_R]$ is empty. But this happens only if the neighborhood $N_{G'}(x)$ of x in G' is a clique in G. In this situation, there maximal cliques of G that intersect $N_{G'}(x)$ form an interval P_G^I in P_G, in

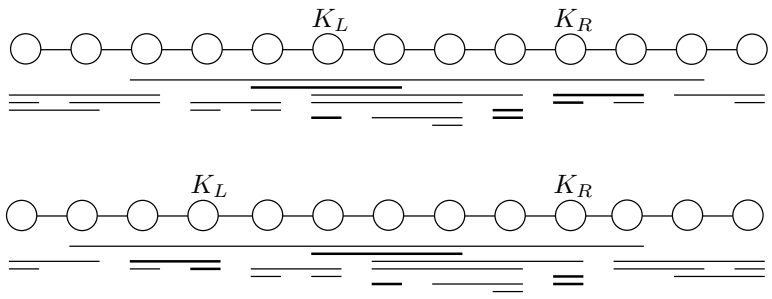


Figure 6.2: Two clique paths of G - different intervals $[K_L : K_R]$.

which every bag contains $N_{G'}(x)$. Therefore, K_L is the rightmost bag of P_G^I and K_R is the leftmost one. The case where $N_{G'}(x)$ is a clique will be treated separately. To put it in more formal words:

Lemma 6.2.3. *Given a non-interval graph G' such that $G = G' - x$, for a fixed $x \in V$, is an interval graph, and a clique path P_G of G, either the neighborhood of x in G' is a clique or K_L and K_R are distinct and define a non-empty interval $[K_L : K_R]$.*

Proof. Suppose the neighborhood of x in G' is not a clique. Then there are distinct maximal cliques of G that contain some neighbors of x. We claim that K_L is strictly to the left of K_R in P_G. Indeed, suppose it is not the case, so K_L is equal K_R or placed to the right of K_R in P_G. Thus, by definition of K_L, every neighbor of x that appears to the left of K_L in P_G is also present in K_L. Analogously, every neighbor of x that appears to the right of K_R in P_G is also present in K_R. Since K_L is to the right of K_R, every neighbor of x belongs to K_R. □

6.2.1 $N_{G'}(x)$ is a clique

If $N_{G'}(x)$ is a clique, it is quite straight forward to compute an minimal interval completion of G'. Therefore, we give the full description of this case here. The next theorem shows that such a completion H can be obtained by filling S, a well chosen minimal separator of G. We need to choose S such that there is a clique path P_G of G with an edge e_S representing S incident to a maximal clique K of G which contains $N_{G'}(x)$. In this way, an interval model of the completion can be obtained by adding in P_G, between the bags incident to e_S, a bag containing $S \cup N_{G'}(x)$, as in the proof of Theorem 6.2.4. Thus, it is enough to fix a clique path P_G and compare, with respect to their intersection with the set of non-neighbors of x in G', the minimal separators incident in P_G to cliques containing $N_{G'}(x)$. The separator S for which this set is inclusion minimal yields an inclusion minimal set of fill edges when S is filled with x. Therefore, a simple comparison of cardinalities gives a linear time algorithm.

Theorem 6.2.4. *Given a non-interval graph G' such that $G = G' - x$, for a fixed $x \in V$, is an interval graph and $N_{G'}(x)$ is a clique. Let H be a minimal interval completion of H with every fill edge incident to x. Then there is a minimal separator S of G such that every fill edge is incident to a vertex in S. Moreover, there is a clique path P_G of G such that there is an edge e_S in P_G which corresponds to S and is incident to a maximal clique of G which contains $N_{G'}(x)$.*

Proof. Let us first prove that there is a unique maximal clique of H that contains x. Let P_H be a clique path of H and let P_G be a clique path of G obtained by pruning x from P_H. Let Ω be

a maximal clique of H that contains x. In fact, Ω is unique. Suppose there are several maximal cliques of H that contain x. W.l.o.g. we may assume that we pick for Ω a bag that contains $N_G[x]$. Let Ω' be another clique that contains x. Remove x from every bag of P_H except for Ω. This yields an intersection model of an interval graph $\overline{H}$ which also is an interval completion of G' with all fill edges incident to x. Moreover, the new graph does not contain the edge xv, for any $v \in \Omega' \setminus \Omega$, which contradicts the minimality of H.

Let us now show that there is a minimal separator S of G that is filled with x in H. Notice that every maximal clique of H, except for Ω, is also a maximal clique of G. Ω is not an endpoint of P_H, since G' is not an interval graph. Indeed, suppose it is the left endpoint of P_H. Then replacing Ω with $N_{G'}[x] - -(\Omega \setminus \{x\})$ in P_H yields an interval model for G'. Which contradicts the fact that G' is not an interval graph. Let K', K'' be the maximal cliques next to Ω in P_H. If Ω contains a maximal clique K of G then K', K are adjacent in P_G, since P_G comes from pruning x from P_H. By Lemma 2.2.20, $S = K' \cap K$ is a minimal separator of G incident in P_G (as the corresponding edge) to K, so $\Omega \subseteq K$, which is a maximal clique containing $N_G(x)$. If Ω does not contain a maximal clique of G, then $\Omega \setminus \{x\}$ is contained in K' or in K''. W.l.o.g. we may assume that $\Omega \setminus \{x\} \subset K'$. Moreover, K', K'' are adjacent in P_G, so again, by Lemma 2.2.20, $S = K' \cap K''$ is a minimal separator of G incident to K', a maximal clique containing $N_G(x)$.

Finally, let us prove that every fill edge is incident to a vertex in the separator S defined above. Suppose there is a fill edge xz not incident to any vertex in S. So Ω contains the vertex z which is only contained in maximal cliques at one side of Ω in P_G. W.l.o.g., let us say that z does not belong to K''. Therefore, we may replace Ω with $\Omega_z - -\Omega_x$ in P_H, where $\Omega_z = \Omega \setminus \{x\}, \Omega_x \setminus \{z\}$, to obtain an interval model of an interval completion of G' which is strictly contained in H. A contradiction with minimality of H. □

We can close the analysis of the case where $N_{G'}(x)$ is a clique with the following corollary. For the complexity, we again use the data structure presented in [9], which permits to analyze all the separators of G in time linear in n.

Corollary 6.2.5. *Let G' be a non-interval graph such that $G = G' - x$, for a fixed $x \in V$, is an interval graph, and the neighborhood of x is a clique. There is an algorithm linear in n which computes a minimal interval completion of G'.*

The rest of the chapter is dedicated to the case where $N_{G'}(x)$ is not a clique.

6.2.2 $N_{G'}(x)$ is not a clique

We are in the case where $N_{G'}(x)$ is not a clique. Let H be a minimal interval completion of G, and let P_G be a clique path of G, such that H respects P_G. By Lemma 6.2.3, the cliques K_L and K_R of P_G define a non-empty interval $[K_L : K_R]$ in P_G. The following theorem describes how to reconstruct H, knowing G' and P_G. Notice that P_G is a nice clique path, an object that the minimal interval completion algorithm will be looking for. Once a nice clique path is found, the following theorem gives the corresponding completion. On its own, this procedure can be used to compute the best interval completion that respects the given clique path.

Lemma 6.2.6. *Let G' be a non-interval graph such that $G = G' - x$, for a fixed $x \in V$, is an interval graph, and the neighborhood of x is not a clique. Let H be a minimal interval completion of G respecting P_G. Then $xu \in E(H) \setminus E(G')$ implies that $u \in K$, for some K strictly between K_L and K_R in P_G. Moreover, every clique K strictly between K_L and K_R in P_G is filled with x in H.*

Proof. Let P_H be a clique path of H such that P_G comes from pruning x from P_H. Let Ω_L be the maximal clique of H that contains K_L. Notice that Ω_L is unique and equals either K_L or $K_L \cup \{x\}$. Let Ω_L^+ denote the maximal clique of H which is just to the right of Ω_L in P_H. The same observations hold for Ω_R, the maximal clique of H that contains K_R. Let Ω_R^- be the maximal clique of H which is just to the left of Ω_R in P_H.

Notice that the bags outside the $[\Omega_L : \Omega_R]$ interval are the same in P_H and P_G. Indeed, suppose it is not the case, so there is x present in some bag outside the $[\Omega_L : \Omega_R]$ interval. W.l.o.g., assume x belongs to some clique to the left of Ω_L in P_H. Let Ω be the leftmost such clique. Let $z \in \Omega$ be a vertex that does not belong to any bag to the right of Ω. It exists, since Ω is a maximal clique. Notice that z is only contained in bags strictly to the left of K_L in P_G. Otherwise, by construction, z would also be present in some bag to the right of Ω_L in P_G. So $z \notin N_{G'}(x)$, since K_L is the leftmost bag of P_G which contains an element of $N_{G'}(x)$ which is not present in any bag to the right of it. Now we may replace Ω with $\Omega_z - -\Omega_x$ in P_H, where $\Omega_z = \Omega \setminus \{x\}, \Omega_x \setminus \{z\}$, to obtain an interval model of an interval completion of G' which is strictly contained in H. A contradiction with minimality of H.

By this minimality argument, $\Omega_L = K_L \cup \{x\}$ implies that $\Omega_L \setminus \Omega_L^+ \subseteq N_{G'}(x)$, and $\Omega_R = K_R \cup \{x\}$ implies that $\Omega_R \setminus \Omega_R^- \subseteq N_{G'}(x)$. If $\Omega_L = K_L$ then $\Omega_L^+ = \{x\} \cup (K_L \cap N_{G'}(x)) \cup (K_L \cap K_L^+)$. Similar argument holds for Ω_R and Ω_R^+. So, by definition of a clique path, x is present in every bag strictly between Ω_L and Ω_R in P_H. Thus, every bag K strictly between K_L and K_R in P_G is filled with x. Moreover, every vertex $y \in N_H(x)$ belongs to a bag strictly between K_L and K_R in P_G or is a neighbor of x already in G'. □

Finally, we give a characterization of nice clique paths. It confirms the intuition, that a clique path P should be nice if the interval $[K_L : K_R]$ in P is inclusion minimal over all clique paths of G.

Theorem 6.2.7. *Let G' be a non-interval graph such that $G = G' - x$, for a fixed $x \in V$, is an interval graph, and the neighborhood of x is not a clique.*

A clique path P_G of G is nice iff the set of maximal cliques of G strictly between K_L and K_R in P_G is inclusion minimal among all clique paths of G.

Proof. Assume P_G^1 is a nice clique path. Let H be a minimal interval completion of G' such that H is respecting P_G^1. Let K_L^1, K_R^1 denote the cliques K_L, K_R of P_G. By Lemma 6.2.6, every fill edge is incident to a vertex in a clique strictly between K_L and K_R in P_G. Suppose there is another clique path P_G^2 of G such that the set of cliques between K_L^2 and K_R^2, where K_L^2, K_R^2 denote K_L, K_R of P_G^2, is strictly included in the set of cliques between K_L^1 and K_R^1 in P_G^1. Then the graph H' corresponding to the interval model obtained by adding x to every bag strictly between K_L^2 and K_R^2 and creating new bags: $K_L^2 \cap N_{G'}(x) \cup S_L^2$ just to the right of K_L^2 and $K_R^2 \cap N_{G'}(x) \cup S_R^2$ just to the left of K_R^2, where S_L^2 (resp. S_R^2) is the separator just to the right (resp. left) of S_L^2 (resp. S_R^2) in P_G^2, is an interval completion of G' strictly included in H. Clearly, H' is an interval completion of g'. Every fill edge in H' is adjacent to a vertex in a bag strictly between K_L^2 and K_R^2 in P_G^2. By definition, it is a strict subset of the bags of P_G^1 that are filled with x in H. Moreover, there is a bag K which is between K_L^1 and K_R^1 in P_G^1, but outside the K_L^2-K_R^2 interval in P_G^2. Since K is a maximal clique, there is a vertex $v \in K$ which is not present in any clique between K_L^2 and K_R^2 (inclusive). v is not a neighbor of x in G', since K_L^2, K_R^2 are the external cliques in P_G^2 that contain a vertex in $N_{G'}(x)$ not contained in any bag strictly to the right, resp. left, of them. So $xv \in E(H) \setminus E(H')$, which contradicts the fact that H is a minimal interval completion of G'.

Conversely, assume that H is an interval completion respecting a clique path P_G^1 as stated in the theorem, with the cliques strictly between K_L and K_R filled with x. It is an interval completion, indeed, by argument described above. Suppose it is not minimal. So, there is H', a minimal interval completion of G' strictly included in H. Let P_G^2 be a clique path of G respected by H', as in Lemma 6.2.6. There is a clique K in the set of cliques strictly between K_L and K_R of P_G^2 which is not in the set of cliques strictly between K_L and K_R of P_G^1, since the latter set is inclusion minimal among all clique paths of G, and the non-equality of these sets is forced by $H' \neq H$. Like above, it means that there is a fill edge in H' not present in H. A contradiction. □

6.3 Minimal completion

We need an algorithm that computes a minimal interval completion in case where the neighborhood of the new vertex is not a clique. For this purpose we use the characterization of Theorem 6.2.7, thus we are looking for a nice clique path. Since the number of clique paths of an interval graph G can be exponential in the number of vertices of G, we need a smart way to browse through this set. For this purpose, we employ the pre-order described in Chapter 5.

In this section we extend the theory of nice clique paths to ordered partitions of G. Recall from Chapter 5 that a minimal separator S of an interval graph defines a partition of the set $\mathcal{K}$ of maximal cliques of G into blocks that appear consecutively in every clique path of G. First, we pick a separator S, which is a minimal u, v-separator for some $u, v \in N_{G'}(x)$. We prove that such a separator appears inside the $[K_L : K_R]$ interval of any clique path of G. Then we find an ordered partition which minimizes the set of blocks inside the interval $[O_L : O_R]$, the analogue of $[K_L : K_R]$, over all valid ordered partitions of G. The general goal is to choose a valid ordered partition OP which minimizes the set of maximal cliques of G that appear between K_L and K_R in any clique path P_G which is a refinement of OP. That gives the first approximation of the set of maximal cliques that will be filled with x in the computed minimal interval completion. Then the parts of OP are recursively refined, like in Theorem 5.1.14, until the set of maximal cliques that will be filled with x is fixed - which completes the computation.

Let $\mathcal{S}_x$ denote the set of minimal separators of G, that are minimal a, b-separators for some $a, b \in N_{G'}(x)$. As announced before, the following theorem states that, for any $S \in \mathcal{S}_x$, S is filled with x in any minimal interval completion of G'. By Lemma 6.2.6, it is equivalent to saying that S appears inside the $[K_L : K_R]$ interval of any clique path of G.

Theorem 6.3.1 ([9]). *Let H be any chordal super-graph of G' such that $H - x = G$. Consider a minimal a, b-separator S of G, where a and b are neighbors of x in G'. Then S is in the neighborhood of x in H.*

6.3.1 Minimal separators in $\mathcal{S}_x$

Let us first generalize the notion of a nice clique path (see Definition 6.2.2) to valid ordered partitions based on $\mathcal{O}(S)$, for a minimal separator $S \in \mathcal{S}_x$. Let us use the same notation for a block $O_i \in \mathcal{O}(S)$ and the corresponding set of maximal cliques contained in O_i.

Let O_L (resp O_R) be the leftmost (resp. rightmost) part of OP such that x has a neighbor appearing in O_L (resp. O_R) but not in any part to the right (resp. left) of it. Notice that this formulation, taken from the definition of K_L, K_R, in case of ordered partitions is equivalent to saying that O_L, O_R are the outermost parts in OP which contain "private" neighbors of x, i.e.

neighbors that are only contained in one block of $\mathcal{O}(S)$. Notice that O_L and O_R of OP are distinct, since $\mathcal{O}(S)$ contains at least two blocks with private neighbors of x (e.g. the one containing a, the other containing b, such that $a, b \in N_{G'}(x)$ and S is a minimal a, b-separator).

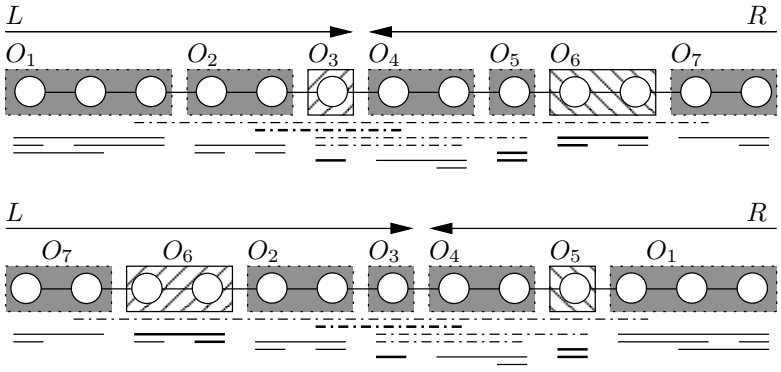


Figure 6.3: Two ordered partitions of G - different intervals $[O_L : O_R]$.

In Figure 6.3, there are two ordered partitions of the same graph G. In both cases, O_L is marked slash pattern, O_R is marked with backslash pattern. Notice that the set of blocks inside the $[O_L : O_R]$ interval of the top example is strictly included in the $[O_L : O_R]$ interval of the bottom example. Here again, we aim to make the interval $[O_L : O_R]$ inclusion minimal over all valid ordered partitions.

The notions generalize to ordered partitions as follows.

Definition 6.3.2. *An interval completion H of G'* respects an ordered partition OP *if it respects a clique path P which is a refinement of OP.*

Definition 6.3.3. *An ordered partition OP is a* nice ordered partition *of the set $\mathcal{K}$ of maximal cliques of G if there is a nice clique path of G which is a refinement of OP. A valid pair (L, R) which corresponds to a nice ordered partition is a* nice pair*.*

It is not difficult to verify that, by Lemma 6.2.6 and Definitions 6.3.2, 6.3.3, the following theorem holds. It describes the relation between a minimal interval completion H of G and any ordered partition OP respected by H.

Theorem 6.3.4. *Let G' be a non-interval graph such that $G = G' - x$, for a fixed $x \in V$, is an interval graph, and the neighborhood of x is not a clique. Let H be a minimal interval completion of G respecting a valid ordered partition OP. Then $xu \in E(H) \setminus E(G')$ implies that $u \in O_i$, for some part O_i between O_L and O_R (inclusive) in P_G. Moreover, every part O_i strictly between O_L and O_R in OP is filled with x in H.*

In other words, given a nice ordered partition OP and a minimal interval completion H of G which respects OP, every block that is inside the $[O_L : O_R]$ interval of OP is filled with x in H, every blocks which is outside this interval stays unchanged, and the blocks O_L, O_R need some extra attention. We will see later, that for them the corresponding graphs $H[O_L \cup \{x\}]$, respectively $H[O_R \cup \{x\}]$, are interval completions which are minimal over all interval completions of $G'[O_L \cup \{x\}]$, respectively $G'[O_R \cup \{x\}]$, that can be extended to a completion of G' respecting

OP. By techniques based on Lemma 5.1.13, we know that such a completion can be computed as a minimal interval completion of the corresponding auxiliary graph $G^+[O_L]$ (resp. $G^+[O_R]$) augmented with the vertex x adjacent to its neighbors in G' and to the dummy vertex. Let us express these ideas in terms of clique paths and ordered partitions.

Like described in Theorem 5.1.14, we compute a nice clique path in a recursive way.

First, we compute a valid ordered partition OP such that the corresponding interval $[O_L : O_R]$ is inclusion minimal. Then, we recursively find ordered partitions of O_L and O_R. For this purpose, we work on the graphs $G^+[O_L]$ and $G^+[O_R]$ (see 5.1.12), where the dummy vertex d is considered to be also adjacent to x. This technique permits to use the same tools for computing a nice ordered partition, as at the first level of the recursion. By Theorem 5.1.14, all clique paths of G are taken into account.

We want to maximize the set of maximal cliques of G which are not filled with x in the completion based on a refinement of OP. In order to correctly evaluate different permutations of $\mathcal{O}(S)$, we need to know for each block O_i if it contains maximal cliques that can be "spared". Roughly speaking, that means that if we manage to put O_i outside the interval $[O_L : O_R]$ in OP, then there is a completion respecting OP in which not the whole O_i is filled with x. We formalize this idea with the following definitions.

Definition 6.3.5. *[see Definition 5.1.12] The graph $G^*[O]$ is obtained from $G'[O]$ by adding the dummy vertex d; d is made adjacent to every element of $S(O)$ and to the vertex x.*

Definition 6.3.6. *Let G' be a non-interval graph such that $G = G' - x$, for a fixed $x \in V$, is an interval graph, and the neighborhood of x is not a clique.*

We say that a block O associated to a minimal separator $S \in \mathcal{S}_x$ is:

- clean *if $O \cap N_{G'}(x) \subset S(O)$,*
- hit *if it is not clean,*
- sparable *if there is an interval completion of $G^*[O]$, with all the fill edges incident to x, in which x is not universal.*

Notice that, following this definition, a clean block is sparable. Moreover, by Lemma 5.1.13, a block is sparable if and only if there is an interval completion $H[O]$ of $G'[O]$ that can be extended to an interval completion of G' with all the fill edges incident to x, such that x is not universal in $H[O]$ (see Lemmas 5.1.8, 5.1.13).

Notice that the sparability can easily be checked in polynomial time by verifying, for every $y \in O$, if the graph obtained from $G^*[O]$ by making x adjacent to every vertex except for y is interval.

With this definition we can describe the set of blocks that, by Theorem 6.3.4, will be filled with x in any completion respecting OP.

Definition 6.3.7. *Given an ordered partition OP based on the partition $\mathcal{O}(S)$ for a minimal separator $S \in \mathcal{S}_x$, let $Filled(OP)$ denote the set of blocks that are strictly between O_L and O_R of OP or in $\{O_L, O_R\}$ and not sparable.*

The following theorem characterizes the structure of a nice clique path in terms of ordered partitions. This is the main result of this chapter. It gives a general description of our approach to computing a nice clique path.

Theorem 6.3.8. *Let G' be a non-interval graph such that $G = G' - x$, for a fixed $x \in V$, is an interval graph, and the neighborhood of x is not a clique.*

A clique path P_G of G is nice iff, for any minimal separator $S \in \mathcal{S}_x$, P_G is a refinement of a valid ordered partition OP based on the partition $\mathcal{O}(S)$ such that:

1. *the set $Filled(OP)$ is inclusion minimal among all ordered partitions based on $\mathcal{O}(S)$,*
2. *the set of maximal cliques strictly after K_L of P_G^L, where P_G^L is the refinement of O_L in P_G, is inclusion minimal among all clique paths of $G[O_L]$ that can be extended into a refinement of OP (see Lemma 5.1.8),*
3. *the set of maximal cliques strictly before K_R of P_G^R, where P_G^R is the refinement of O_R in P_G, is inclusion minimal among all clique paths of $G[O_R]$ that can be extended into a refinement of OP.*

Notice that the second and third condition correspond to the fact that $H[O_L \cup \{x\}]$ (resp. $H[O_L \cup \{x\}]$) is an inclusion minimal completion, over interval completions of $G'[O_L \cup \{x\}]$ (resp. $G'[O_L \cup \{x\}]$) that can be extended to a completion of G respecting OP.

Proof. Assume that P is nice. Let S be any minimal separator in $\mathcal{S}_x$ and let OP be the partition $\mathcal{O}(S)$ ordered as the corresponding cliques appear in P. Recall that, by Theorem 6.2.7, the set of maximal cliques strictly between K_L and K_R of P_G is inclusion minimal among all clique paths of G. In particular, the set of blocks associated to S that appear entirely between K_L and K_R (inclusive) in P_G is inclusion minimal among all clique paths of G. Hence the set $Filled(OP)$ is also inclusion minimal among all ordered partitions based on $\mathcal{O}S$, for any OP such that P_G is a refinement of OP. The set of maximal cliques strictly after K_L of P_G^L is inclusion minimal among all clique paths of $G[O_L]$ with $S(O_L)$ contained in the last maximal clique. Indeed, suppose there is $P_G^{'L}$, a clique path of $G[O_L]$ with $S(O_L)$ contained in the last clique, with a strictly smaller set of maximal cliques strictly after K_L' of $P_G^{'L}$. So, by Lemma 5.1.8, we can replace P_G^L with $P_G^{'L}$ in P_G and obtain a clique path with strictly smaller set of maximal cliques between K_L and K_R - a contradiction. An analogous argument holds for P_G^R and the conclusion follows.

Conversely, assume that, for any minimal separator $S \in \mathcal{S}_x$, P_G is a refinement of a valid ordered partition OP based on $\mathcal{O}(S)$, that satisfies the conditions enumerated in the theorem. Suppose that P_G is not nice, so the set of maximal cliques of G strictly between K_L and K_R of G is not inclusion minimal among the clique paths of G. Let $P_G^{'}$ be a clique path of G for which this set is inclusion minimal and is strictly contained in the corresponding set for P_G. The set of blocks associated to S that appear entirely between K_L and K_R (inclusive) in P_G is inclusion minimal among all clique paths of G, so it is the same in $P_G^{'}$. The difference lies in O_L or O_R of OP. Assume, w.l.o.g., that there is a clique $K \in O_L$, where O_L is the block that contains the cliques of O_L, such that K is strictly between K_L and K_R of P_G but outside K_L' and K_R' of $P_G^{'}$. The cliques of O_L are consecutive also in $P_G^{'}$, so the set of cliques after K_L' in $P_G^{'L}$, where $P_G^{'L}$ is the clique path of $G[O_L]$ in $P_G^{'}$, is strictly smaller than in P_G^L. Moreover, $S(O_L)$ appears in the last clique of $P_G^{'L}$ as well. A contradiction. □

From the above theorem we can extract a characterization of nice ordered partitions.

Corollary 6.3.9. *Let G' be a non-interval graph such that $G = G' - x$, for a fixed $x \in V$, is an interval graph, and the neighborhood of x is not a clique. Let $S \in \mathcal{S}_x$. A valid ordered partition OP*

based on $\mathcal{O}(S)$ is nice iff the set $Filled(OP)$ is inclusion minimal among all valid ordered partitions based on $\mathcal{O}(S)$.

Based on this characterization, in next section, we give an algorithm computing a nice ordered partition.

6.4 Algorithm NicePair

Let us show with an example that the pre-order is not enough to control nice ordered partitions. Let G' be a non-interval graph such that $G = G' - x$, for a fixed $x \in V$, is an interval graph, and the neighborhood of x is not a clique. Let $S \in \mathcal{S}_x$. Let O_1, O_2, O_3 be blocks associated to S that are equivalent with respect to $\preceq$. Let O_1 be clean, O_2 hit but sparable, and O_3 not sparable (see Definition 6.3.6). Assume that they all have to appear in L. If $O_i s$ appear in (L, R) as $O_1 - -O_2 - -O_3$, then $O_L = O_2$. Moreover, by Theorem 6.3.4, in any interval completion H of G that respects P_G, O_1 stays clean and O_2 is not filled with x. Any other configuration of $O_i s$ leads to extra fill edges present in the corresponding interval completion.

Let $(\mathcal{O}(S), \leq)$ denote the pre-order $(\mathcal{O}(S), \preceq)$ refined to incorporate this differentiation.

Definition 6.4.1. *For any $O_i, O_j \in \mathcal{O}(B)$,*
$O_i \leq O_j$ if $O_i \preceq O_j$ and (not $O_j \preceq O_i$ or $priority(O_i) \leq priority(O_j)$), where

$$priority(O) = \begin{cases} 1, & \text{if } O \text{ is clean;} \\ 2, & \text{if } O \text{ is hit but sparable;} \\ 3, & \text{if } O \text{ is not sparable.} \end{cases}$$

The algorithm **NicePair** of Figure 6.4 produces a nice pair (L, R). The algorithm relies on the pre-order $\leq$. We transform it into a linear order $\leq_{top}$ by sorting the blocks in topological order (the permutation of elements in the same equivalence class does not affect the minimality of interval completions obtained). The graph $(\mathcal{O}, \preceq)$ together with a topological ordering $\leq_{top}$ of the refined pre-order is taken as input.

The algorithm processes the blocks of $\mathcal{O}(S)$ one-by-one, given in a topological order, to yield a nice pair (L, R). Initially, all blocks are marked "L or R", what means that they can be put in L or in R. When the algorithm decides to put a block O in one of the lists, this may force some other blocks to be placed in the same or the opposite side, because of the incomparabilities with respect to the pre-order. Recall that if two blocks are incomparable, with respect to $\preceq$, then they have to be put on the opposite sides of the separator S. The procedure **shake** ensures this forcing, by marking with "L" or "R", respectively, each block that is forced to be in the corresponding list by the choice the algorithm made for the current block O.

The algorithm has three phases. During the first one, all considered blocks are clean. They are placed in L or R according to the forcing. Anyway, they appear before O_L or O_R in the corresponding lists, so they will not have any fill edges added. When the first hit block is encountered, the algorithm enters the second phase. The list to which the first hit block is put is marked as *MinHitSide*. Suppose, w.l.o.g., that this list is L. During this phase, we try to put each sparable block to R. That is to say, if the current block O is marked "L or R" then we put it to R, since otherwise the choice is forced. In this way, we prevent some fill edges from being added, since putting O above O_L results in making all its vertices adjacent to x in the corresponding completion. On the other hand, if O is not sparable, then we put it in L. This way we keep the possibility

of sparing some other blocks. The third phase starts when both lists contain hit blocks. A this point, whatever choice we make, all further blocks will be filled.

Notice that choosing the topological order and coloring the graph $(\mathcal{O}(S), \|)$ explores all the freedom we have for creating a nice ordered partition. We will first prove that the topological order chosen is without impact for the nice property of an ordered partition and then that our algorithm gives a nice ordered partition.

Theorem 6.4.2. *For any topological order $\leq_{top}$ there is a nice ordered partition $(L_{\leq_{top}}, R_{\leq_{top}})$ respecting it, in the sense that the two lists are suborders of $\leq_{top}$.*

Proof. Let (L, R) be a nice pair corresponding to a topological order. Permute two elements O_1 and O_2 in this topological order. Let us construct a nice ordered partition (L', R') respecting the new topological order.

If O_1 and O_2 are incomparable w.r.t. $\leq$, then $(L', R') = (L, R)$.

Let $\mathcal{O}_{eq}$ be a maximal set of blocks equivalent for $\leq$, containing O_1, O_2. Then L' and R' are obtained from (L, R) by exchanging O_1 and O_2 (they may be in the same list or in different lists). Suppose that (L', R') is not nice. That means there is a block O that can be put outside the interval $O_{L'}$-$O_{R'}$ and spared.
Suppose $O \notin \mathcal{O}_{eq}$. Then O can be put outside the corresponding interval in (L, R) too. Indeed, the constraints coming from $\leq$ are the same, no matter what is the permutation of blocks $\mathcal{O}_{eq}$. Moreover, by Lemma 5.2.5 the placement of blocks $O \notin \mathcal{O}_{eq}$ relative to O_L and O_R is the same in (L, R) and in (L', R').
Suppose O belongs to $\mathcal{O}_{eq}$. If all $\mathcal{O}_{eq}$ belong to the same connected component of $(\mathcal{O}(S), \|)$ then they form an interval of blocks at the same side of B in every ordered partition. Say that there are in L. If O_L is bigger or smaller than $\mathcal{O}_{eq}$ then a transposition does not change the set $Spared(L, R)$. So $O_L \in \mathcal{O}_{eq}$ and all elements of $\mathcal{O}_{eq}$ are hit but sparable. Since only one block of $\mathcal{O}_{eq}$ can be spared in such setting, we have that $O = O_{L'}$ - a contradiction.
So the blocks of $\mathcal{O}_{eq}$ form singleton connected components of $(\mathcal{O}(S), \|)$. The number of blocks among $\mathcal{O}_{eq}$ that can be spared depends on the number of sides with hit blocks smaller than O_{eq} in (L, R). This maximum is achieved in (L, R), thus in (L', R') too. If O_{eq} is clean (then all or no block in $\mathcal{O}_{eq}$ is in $Spared(L, R)$) or O_{eq} is hit but sparable then O is already spared in interval completions good (L', R'). If O_{eq} is not sparable then O cannot be spared. Anyway, we reach a contradiction. □

In the proof we will show that at any step of the algorithm we obtain a "partial" nice pair (L, R). It is characterized by Corollary 6.3.9, only that the set of blocks considered is restricted to the ones already partitioned into (L, R).

Theorem 6.4.3. *Given graph $(\mathcal{O}(S), \leq)$ for G and B, the algorithm creates a nice pair (L, R).*

Proof. Let us see the algorithm in terms of connected components $C_1, \ldots, C_k$ of $(\mathcal{O}(S), \|)$. They are processed in topological order coming from the topological order of blocks in $\mathcal{O}(S)$. Each time the algorithm finds a block with a label "L or R", it is the minimal block of a new component encountered. The choice of putting it in one side decides, through forcing, about the partitioning of the whole component. Thus (L, R) can be seen as a function assigning to each connected component of $(\mathcal{O}(S), \|)$ a binary value 1 if the first color class is put into L and 0 - otherwise. Let us call it a *binary representation*. Let $\mathcal{O}_1(C_i), \mathcal{O}_2(C_i)$ denote the two color classes of C_i and suppose w.l.o.g. that the minimum element of C_i, with respect to $\leq_{top}$, is in $\mathcal{O}_1(C_i)$.

`procedure` **NicePair**
`Input:` $(\mathcal{O}(S), \preceq, \leq_{top})$ the pre-order on the set of all blocks together with a topological order of the refined pre-order
`Output:` (L, R) - lists of blocks to be constructed s.t. (L, R) is a nice pair
`Variables:` M - an array used to store forcing information on the blocks.

$$M[O] = \begin{cases} \text{“}L\text{”} & \text{- } O \text{ is forced to be in } L; \\ \text{“}R\text{”} & \text{- } O \text{ is forced to be in } R; \\ \text{“}L \text{ or } R\text{”} & \text{- O can be either in } L \text{ or in } R. \end{cases}$$

$MinHitFound$ - information if the minimal hit block has been found
$MinHitSide$ - information on the side where the minimal hit block has been put, initialized to be L but R would be fine as well

`procedure` **shake**(O)
`forall` $X \in \mathcal{O}$ s.t $(O \leq_{top} X)$ and (X is incomparable with O for $\preceq$) and $(M[X] =$ “L or R”)
    $M[X] := opposite(M[O])$ `do`
    shake(X)
`end`

$MinHitFound := false$
$MinHitSide := L$
$L \leftarrow \emptyset; R \leftarrow \emptyset$
`forall` $O \in \mathcal{O}$ `do`
    $M[O] :=$ “L or R”
`forall` $O \in \mathcal{O}$ in the topological order `do`
    `if` ($M[O] =$ “L or R”) `then`
        `if` O is sparable `then`
            move O on the top of $opposite(MinHitSide)$ and $M[O] := opposite(MinHitSide)$
            shake(O)
        `else`
            move O on the top of $MinHitSide$ and $M[O] := MinHitSide$
            shake(O)
        `endif`
    `else`
        move O on the top of $M[O]$
    `endif`
    `if` (not $MinHitFound$ and O is hit) `then`
        $MinHitFound := true$
        $MinHitSide := M[O]$
    `endif`
`endforall`
`return` (L, R)

Figure 6.4: Main case: the algorithm constructing a nice pair (L, R).

We prove that at every step the algorithm creates (L_i, R_i) from (L_{i-1}, R_{i-1}) by choosing to add one color class of C_i to L_{i-1} and the second to R_{i-1} in such a way that the set $Spared(L_i, R_i) \setminus Spared(L_{i-1}, R_{i-1})$ is inclusion maximal among the two possible choices. Creating (L, R) can be divided into three phases. During the first phase, none of the lists contains hit blocks. For the second phase only one list has blocks, and in the third phase both lists have hit blocks.

1. As long as the components $C_1, \ldots, C_{i-1}$ do not contain any hit block, we add $\mathcal{O}_1(C_i)$ to L and $\mathcal{O}_2(C_i)$ to R. Clearly, this decision spares the same set of blocks as the other, thus we have maximality. This stage stops when we encounter C_i with a hit block. Then we fix the `MinHitSide` to point the side containing the first encountered hit block, w.l.o.g. assume that "L" is the side and O_L is the block. If "R" does not contain a hit block then we pass to the step 2, which lasts as long as only "L"contains hit blocks. If "R" contains a hit block mark O_R and pass to the step 3, when both lists contain hit blocks.

2. Recall that at this stage only the left list contains hit blocks. For each C_i we add $\mathcal{O}_1(C_i)$ to R if and only if its minimum element O_i is sparable. If O_i is sparable, our choice ensures that $O_i \in Spared(L_i, R_i)$, while the other choice would imply $O_i \notin Spared(L_i, R_i)$. If O_i is not sparable, by our choice we spare the minimum element O_i' in the opposite color class of C_i (if the latter is sparable). If O_i' is not sparable, either choices would imply $Spared(L_i, R_i) = Spared(L_{i-1}, R_{i-1})$. We have maximality again. This stage stops when we put a hit block in "R" and pass to the step 3.

3. We just add $\mathcal{O}_1(C_i)$ to L and $\mathcal{O}_2(C_i)$ to R. Whatever the decision is, no more blocks can be spared in this step.

So at each step the algorithm chooses an inclusion maximal augmentation of the set of spared blocks. Now let (L, R) be the nice pair obtained by the algorithm. Suppose there is (L', R') with $Spared(L, R) \subset Spared(L', R')$. (L', R') can not be obtained from (L, R) by permutation of elements without changing the sides. As in the proof of Theorem 6.4.2, permutations of elements equivalent w.r.t. the refined pre-order does not increase the set *Spared*. For the sake of simplicity of reasoning but w.l.o.g., we can assume that there are no components equivalent for $\leq$, thus the topological ordering of components is unique.

Take (L', R') to be one sharing with (L, R) the longest prefix of the binary representation among the ordered partitions having the properties described above. Let C be the first, in topological order, component partitioned in (L', R') in another way than in (L, R). We can assume that C is also the fist component that contains more spared blocks in (L', R') than in (L, R) - otherwise we could partition this and all bigger components the other way obtaining (L'', R'') with $Spared(L'', R'') = Spared(L', R')$ and sharing longer prefix with (L, R). That means that for C the algorithm did not choose the twist adding an inclusion maximal set of blocks to the set of spared ones. A contradiction. □

6.5 Putting everything together

Knowing a nice ordered partition OP, we need to process recursively the external blocks O_L, O_R of OP that have private neighbors of x. By Theorem 6.3.8, we need to find clique paths of the corresponding graphs $G[O_L]$, $G[O_R]$ that minimize the set of cliques between K_L and K_R over the

refinements of OP. We prove that this problem is equivalent to finding nice ordered partitions for the corresponding auxiliary graphs.

Recall that, analyzing $G^+[O]$, we consider the dummy vertex d to be adjacent to x.So we analyze the graph $G^*[O]$ obtained from $G'[O]$ by adding the dummy vertex d; d is made adjacent to every element of $S(O)$ and to the vertex x. Moreover, by Lemma 5.1.13, every clique path of $G^+[O]$ is of the form $P - -(\{d\} \cup S')$.

The following lemma describes the way we use the auxiliary graph $G^+[O]$ to find a clique path $P_{G[O]}$ of $G[O]$ that satisfies the corresponding condition of Theorem 6.3.8.

Lemma 6.5.1. *Let G' be a non-interval graph such that $G = G' - x$, for a fixed $x \in V$, is an interval graph, and the neighborhood of x is not a clique. Let $S \in \mathcal{S}_x$.*

Let O be a block associated to S, $S' = S \cap O$. The set of maximal cliques strictly after K_L of $P_{G[O]}$ is inclusion minimal among all clique paths of $G[O]$ such that the rightmost clique in $P_{G[O]}$ contains S' iff $P_{G^+[O]}$, where $P_{G^+[O]} = P_{G[O]} - -(\{d\} \cup S')$, is a nice clique path of $G^+[O]$, with respect to $G^*[O]$.

Proof. This is a straightforward consequence of Lemma 5.1.13 and Theorem 6.2.7. □

At some level of recursion, it may happen that $S = S(O)$ is the only separator in $\mathcal{S}_x$ of the auxiliary graph $G^+[O]$. In this situation, we cannot again choose S to be used in the partitioning algorithm. But the following lemma shows that this situation can be handled in a similar way.

Lemma 6.5.2. *Let G' be a non-interval graph such that $G = G' - x$, for a fixed $x \in V$, is an interval graph, and the neighborhood of x is not a clique. Let $S \in \mathcal{S}_x$ and let O be a block associated to S such that S is the only separator in $\mathcal{S}_x$ present in $G^*[O]$ (the neighborhood of x in O is a clique).*

Let B be the union of maximal cliques of $G[O]$ that contain $N_{G'}(x) \cap O$. Consider the minimal separators of G contained in O that contain S and have an associated full component contained in $V \setminus O$. Pick T to be inclusion maximal among them. Let O_B be the element of $\mathcal{O}(T)$, the partition of maximal cliques of $G^+[O]$ into blocks associated to T, which contains B. Let O_d be the element of $\mathcal{O}(T)$ that contains d. Any ordered partition based on $\mathcal{O}(T)$ with inclusion minimal set of blocks strictly between O_B and O_d is nice. Moreover, any refinement of such an ordered partition is a nice clique path of $G^+[O]$.

Proof. S is the only minimal u, v-separator for some u, v in the neighborhood of x in $G^*[O]$, so S is contained in every maximal clique in B. Moreover, no other clique of G contains the whole $N_{G'}(x)$, so the cliques contained in B are consecutive in every clique path of G (and the same holds for $G^+[O]$). Take any clique path $P_{G^+[O]}$ of $G^+[O]$, where w.l.o.g. $K_d = \{d\} \cup S$ is the rightmost clique (see Lemma 5.1.13). The rightmost edge of $P_{G^+[O]}$ incident to a clique in B corresponds to a minimal separator of $G^+[O]$ contained in B, which contains $N_{G'}(x)$ - so T is well defined.

Consider the blocks $\mathcal{O}(T)$ associated to T in $G^+[O]$. The maximal clique K_d constitutes a block, let us call it O_d, since $S \subseteq T$. The block associated to T that contains maximal cliques in B is unique, let us denote it O_B. Indeed, the cliques in B all contain $N_{G'}(x)$ which is not contained in any maximal clique out of B and thus, by Lemma 2.2.20, $N_{G'}(x) \not\subseteq T$. Like in the argument on the clique paths of $G^+[O]$ above, we may assume w.l.o.g. that O_d is the rightmost block in any ordered partition OP of $G^+[O]$ based on T. So O_d is the rightmost block (O_R) of OP which

contains private neighbors of x. Moreover, O_B is the leftmost such block, since there are no other blocks containing private neighbors of x.

Let us take OP to be any such ordered partition which minimizes the set of blocks strictly between O_L and O_R. Let $P_{G^+[O]}$ be a refinement of OP. Clearly, the subpath $P^R_{G^+[O]}$ of $P_{G^+[O]}$ induced by cliques in O_R yields an inclusion minimal set of bags before K_R among all clique paths of $G[O_R]$ extendable to a refinement of OP, since O_R consists of a single clique. Let us show that the same holds for $P^L_{G^+[O]}$, which together with the minimality of the set of blocks in $\mathcal{O}(T)$ contained entirely between K_L and K_R in $P_{G^+[O]}$ yields that $P_{G^+[O]}$ is nice (see the proof of Theorem 6.3.8). Notice that Theorem 6.3.8 cannot be used here directly, since $T \notin \mathcal{S}_x$.

If the blocks on both sides of O_B contain T, then $O_B = B$. Otherwise, there would be a maximal clique K' of $G^+[O]$ containing T, contained in O_B but not in B, adjacent in $P_{G^+[O]}$ to a clique in B. This would imply the existence of the minimal separator $K' \cap B$ strictly containing T, which contradicts the choice of T. And $O_B = B$ implies that all bags of O_L contain the same neighbors of G and thus, K_L is the rightmost clique in the subpath of $P_{G^+[O]}$ induced by cliques in O_L. So the set of cliques strictly after K_L is inclusion minimal.

Now consider the case where only at one side of O_B there is a block that contains T, let it be O_T. By argument used above, there is no maximal clique contained in O_B but not in B that contains T. So the endpoint clique of the subpath $P^L_{G^+[O]}$ of $P_{G^+[O]}$ induced by cliques in O_B is in B. K_L is the rightmost bag contained in B. Since the cliques of B are consecutive in every clique path of $G^+[O]$, the positions relative to K_L in $P^L_{G^+[O]}$ are determined by the placement of O_T : either all bags contained in O_B but not in B are after or all are before K_L. Since all feasible permutations of bags in O_B yield the same set of bags strictly after K_L, it is always minimal. The conclusion follows. □

All the discussion on nice ordered partitions can be summarized with the following theorem, which is a direct consequence of presented claims.

Theorem 6.5.3. *Let G' be a non-interval graph such that $G = G' - x$, for a fixed $x \in V$, is an interval graph, and the neighborhood of x is not a clique. Let $S \in \mathcal{S}_x$.*

Any clique path of G which is a refinement of an ordered partition OP' obtained from a nice ordered partition OP based on $\mathcal{O}(S)$, by recursively computing nice ordered partitions of $G^+[O_L]$ and $G^+[O_R]$ based on a minimal separator in $\mathcal{S}_x \setminus \{N_G(O_L)\}$ (resp. $\mathcal{S}_x \setminus \{N_G(O_R)\}$), where O_L (resp. O_R) is the block corresponding to O_L (resp. O_R) of OP, until the neighborhood of x in O_L (resp. O_R) is a clique and we use Lemma 6.5.2 to solve this base step of the recursion, is a nice clique path of G.

6.5.1 Algorithm MinimalIntervalCompletion

Given an interval graph $G = (V, E)$ and a vertex x outside of G, we run the following procedure with $(G, x, N_{G'}(x))$ as input. A minimal interval completion of G' is obtained as $H' = (V', E')$ with $V' = V \cup \{x\}$ and $E' = E \cup \{\{x, y\} \mid y \in \textbf{IntervalCompletion}(G, x, N_{G'}(x))\}$.

We choose a separator $S \in \mathcal{S}_x$ and then call the algorithm **NicePair** to obtain the nice pair (L, R). In order to compute a minimal interval completion of G', according to Theorem 6.5.3 it remains to compute, by recursive calls, minimal interval completions of $G'_d[O \cup \{x\}]$ for each $O \in \{O_L, O_R\}$.

`procedure` **MinIntervalCompletion**

`input:` G, x, N - the neighborhood of x in G', d - dummy vertex
`output:` modified N - the neighborhood of x in an interval completion of G'

`if` G' is an interval graph `then`
  `return` $\emptyset$

Compute $\mathcal{S}_x$.
`if` $N(d)$ is not the only minimal separator in $\mathcal{S}_x$ `then`
  $S :=$ a minimal separator in $\mathcal{S}_x$
  $N := N \cup S$
`else`
  $S :=$ a minimal separator chosen like in Lemma 6.5.2
`endif`
Compute the blocks $\mathcal{O}(S)$
`foreach` $O \in \mathcal{O}(S)$ `do`// compute minimal interval completions of auxiliary graphs
  let $G_d[O] := G[O]$ plus a dummy vertex d adjacent to $O \cap S$
  $N := N \cup$ **MinIntervalCompletion**$(G_d[O], x, N \cap O \cup \{d\}$,d$) \setminus \{d\}$
`forall` O in $\mathcal{O}(S)$ compute *priority*(O) // based on the completion computed above
Compute the graph $(\mathcal{O}(S), \leq)$ and a topological order $\leq_{top}$
$(L, R) \leftarrow$ **NicePair**$((\mathcal{O}(S), \leq, \leq_{top}))$
`forall` O in L above O_L `do` $N := N \cup O$
`forall` O in R above O_R `do` $N := N \cup O$
`return` N

Theorem 6.5.4. *Algorithm* **MinIntervalCompletion** *computes a minimal interval completion of G' in $O(n^2 \log n)$ time.*

Proof. We show that one call of the procedure costs $\mathcal{O}(n' + m')$ time, where n', m' denote the number of vertices and the number of edges of the processed subgraph. As the number of edges in a completion may grow up to n^2, one level of recursion takes in total $\mathcal{O}(n^2)$ time. The information on sparability is taken from lower levels of recursion, based on the information if x is a universal vertex in the interval graph computed at the bottom level. In every call, the separators used for subsequent calls are chosen in $\mathcal{S}_x$ such that they partition $\mathcal{S}_x$ in two subsets of roughly half the size. In this way we ensure at most $\log n$ recursion levels. □

Theorem 6.5.5. *There is an algorithm computing a minimal interval completion of an arbitrary graph in $O(n^3 \log n)$ time.*

Proof. Let $\{x_1, \ldots, x_n\}$ be an arbitrary ordering of the vertices of G. We compute incrementally a minimal interval completion H_i of $G_i = G[\{x_1, \ldots, x_i\}]$, respecting H_{i-1}. For this purpose we use Corollary 6.2.5 if the neighborhood of x_i in G_{i-1} induces a clique in H_{i-1} and Theorem 6.5.4 otherwise. □

6.6 Conclusions

We gave in this chapter a $\mathcal{O}(n^3 \log n)$ time algorithm computing a minimal interval completion of an arbitrary input graph. It is an incremental algorithm, that can work on any ordering $(v_1, \ldots, v_n)$

of $V(G)$. At step i, it computes a minimal interval completion H_i of $G_i = G[\{v_1, \ldots, v_i\}]$, from a minimal interval completion H_{i-1} of G_{i-1}, by only adding fill edges incident to the new vertex v_i. This feature makes the algorithm particularly interesting in dynamic applications, where the considered graph is constructed incrementally, adding vertices one-by-one, and the graph computed so far should not be modified. Within this framework, a little modification of algorithm permits to obtain minimum interval completions or interval completions of minimum clique number. That is to say, given a graph G' (an interval graph G augmented with a vertex x adjacent to a subset of $V(G)$), the algorithm can compute solutions to the following two problems. First: compute a minimum cardinality set F of fill edges incident to x, such that adding F to G' yields an interval graph. The other: compute a set of fill edges F incident to x, such that adding F to G' yields an interval graph of minimum clique number, over such interval completions of G'. Thus, our algorithm can solve a kind of on-line versions of profile and pathwidth problems.

We have found that the ordering of vertices used in the algorithm has great impact on the quality of the completion (off-line version). However, we do not know how to control this feature. In other words, we do not have any characterizations of orderings which yield completions of small pathwidth or small profile. This is one of the directions to explore.

Another direction of prospective research is the pre-order and its relation with path decompositions. As we mentioned earlier, there is a greedy algorithm that uses minimal separators to partition a graph in order to compute a tree decomposition of small width. It seems that the pre-order captures what is needed for an analogous algorithm computing path decompositions of small width. It would be interesting to try creating approximation algorithms for pathwidth on particular graph classes based on the pre-order decomposition.

Chapter 7

Pathwidth of Circular-Arc Graphs

Contents

The pathwidth of a graph G is the minimum clique number of H minus one, over all interval super-graphs H of G. Although pathwidth is a well-known and well-studied graph parameter, there are only very few graph classes for which this parameter is polynomial time tractable. Polynomial algorithms computing pathwidth exist for the class of trees [41], and more generally, for the class of graphs of bounded treewidth [18]. There are also results for graph classes, for which pathwidth is equal to treewidth, e.g. permutation graphs [19]. Actually, all the computational results on pathwidth were based on treewidth. We give in this chapter the first polynomial time algorithm computing pathwidth of circular-arc graphs. Pathwidth of these graphs can be easily approximated within a factor of 2. Nevertheless, for circular-arc graphs, pathwidth is not equal to treewidth, and clearly this class is not of bounded treewidth. Therefore we could not use any of the techniques for computing pathwidth known before. Our algorithm is based on a study of interval completions of circular-arc graphs. We characterize a subclass of these interval completions, containing optimal solutions to the pathwidth problem. Based on this combinatorial result, we give an $O(n^2)$ algorithm computing the pathwidth of circular-arc graphs.

7.1 Folding

The folding is a new tool that proved very useful for general, polynomial time checkable characterization of minimal interval completion (presented in Chapter 8) and for understanding the pathwidth problem on circular-arc graphs. We present the application to pathwidth of circular-arc graphs first, in order to gently introduce the reader to the folding, which is intensively used later, in Chapter 8.

Given a path decomposition P of G, let $\texttt{Fill}(G, P)$ be the graph obtained by adding edges to G so that each bag of P becomes a clique. It is straight forward to verify that $\texttt{Fill}(G, P)$ is an interval

super-graph of G, for every path decomposition P. Moreover P is a clique path decomposition of $\texttt{Fill}(G, P)$.

Recall that an edge clique cover $\mathcal{X}$ of a graph G is a set of (not necessarily maximal) cliques of G, such that each edge of G is contained in at least one of the cliques in $\mathcal{X}$.

Definition 7.1.1 (see also [72])**.** *Let $\mathcal{X}$ be an edge clique cover of an arbitrary graph G and let $\mathcal{Q} = (Q_1, ..., Q_k)$ be a permutation of $\mathcal{X}$. We say that $(G, \mathcal{Q})$ is a* folding *of G by $\mathcal{Q}$. To every folding $(G, \mathcal{Q})$ we associate, by Algorithm* `FillFolding` *of Figure 7.1, the completion $H =$ $\texttt{FillFolding}(G, \mathcal{Q})$* defined *by the folding $(G, \mathcal{Q})$.*

The algorithm `FillFolding` takes as input an arbitrary graph G and a sequence $\mathcal{Q} = (Q_1, ..., Q_k)$ of subsets of $V(G)$. The algorithm modifies $\mathcal{Q}$, adding vertices to some bags in $\mathcal{Q}$ in order for the set of bags containing x to be contiguous. For each vertex x of G the leftmost bag Q_l and the rightmost bag Q_r in $\mathcal{Q}$ containing x are computed, and x is added to every bag in the interval $[Q_l : Q_r]$. Finally, with the operator `Fill`, these new bags are turned into cliques by adding the necessary fill edges. The resulting completion is denoted $H = \texttt{FillFolding}(G, \mathcal{Q})$.

Algorithm `FillFolding`
Input: Graph $G = (V, E)$ and $\mathcal{Q} = (Q_1, ..., Q_k)$, a sequence of subsets of V;
Output: A super-graph H of G;

$P = \mathcal{Q}$;
for each vertex v of G **do**
  $s = \min\{i \mid x \in Q_i\}$;
  $t = \max\{i \mid x \in Q_i\}$;
  for $j = s + 1$ to $t - 1$ **do**
    $P_j = P_j \cup \{v\}$;
end-for
$H = \texttt{Fill}(G, P)$;

Figure 7.1: The `FillFolding` algorithm.

Lemma 7.1.2. *Given a folding $(G, \mathcal{Q})$ of G, the graph $H =$ $\texttt{FillFolding}(G, \mathcal{Q})$ is an interval completion of G.*

Proof. Observe that after the **for** loops, P is a path decomposition of H, since every edge is contained in some bag, and for every vertex the bags containing it induce a subpath of P. Hence, since $H = \texttt{Fill}(G, P)$, it is an interval completion of G. □

The graph defined by a folding is not necessarily a minimal interval completion of G. Nevertheless, we prove in Theorem 7.1.3 that every minimal interval completion of G is defined by some folding.

Theorem 7.1.3. *Let H be a minimal interval completion of a graph G with an edge clique cover $\mathcal{X}$. Then there exists a folding $(G, \mathcal{Q})$, where $\mathcal{Q}$ is a permutation of $\mathcal{X}$, such that $H =$ $\texttt{FillFolding}(G, \mathcal{Q})$.*

Proof. Let $\mathcal{X} = \{X_i \mid 1 \leq i \leq p\}$ and $\mathcal{K} = \{\Omega_i \mid 1 \leq i \leq k\}$ denote an enumeration of $\mathcal{X}$ and the set of maximal cliques of H, respectively. Let $P = (\Omega_1, \ldots, \Omega_k)$ be a clique-path of H. It defines a linear order on the set $\mathcal{K}$. Let us use it to construct a linear order on $\mathcal{X}$.

In a natural way, P defines a linear pre-order on $\mathcal{X}$ by

$$X_a \leq X_b \text{ if } \exists i, j \text{ such that } X_a \subseteq \Omega_i, X_b \subseteq \Omega_j, \text{ where } 1 \leq i \leq j \leq k, 1 \leq a, b \leq p,$$

where for a clique X_i that is contained in several maximal cliques of H, consider just the first occurrence. Transform it into a linear order (sequence) $\mathcal{Q}$ by fixing any permutation inside the equivalence classes.

Let us define $H' = \texttt{FillFolding}(G, \mathcal{Q})$, and prove that $H' = H$. By Lemma 7.1.2, H' is an interval completion of G. Moreover, $E(H') \subseteq E(H)$, since $xy \in E(H')$ only if the interval between the first and the last element in $\mathcal{Q}$ that contains x intersects the one corresponding to y. In this case, the corresponding intervals in P intersect as well, so there is $xy \in E(H)$. By minimality of H, there is $H = H'$. □

It is well-known (see also Definition 2.3.2) that the pathwidth of G is the minimum, over all interval completions H of G, of the clique number of H minus one. Clearly, we can restrict to minimal interval completions. Theorem 7.1.3 tells us that the optimal interval completion for the pathwidth problem is defined by some folding of the graph.

7.2 Circular-arc graphs

Recall that G is a circular-arc graph if it is the intersection graph of a family of arcs on a cycle. Also, by Lemma 2.2.10, a circular-arc graph G has a clique cycle representation $(\mathcal{X}, C)$.

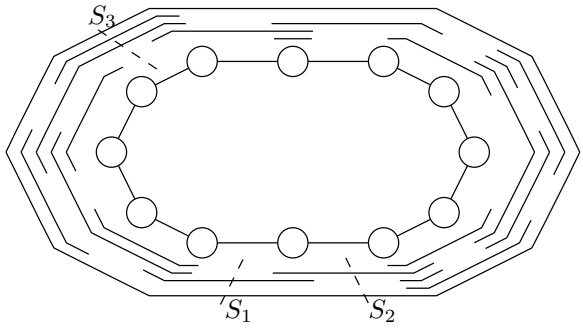


Figure 7.2: Circular-arc graph.

Definition 7.2.1. *Given a clique cycle $(\mathcal{X}, C)$ of a circular-arc graph, each edge of $(\mathcal{X}, C)$ represents the intersection of the incident cliques, and is called a* semi-separator *of G.*

In Figure 7.2, there is a clique cycle of a sample circular-arc graph G. Arcs representing the vertices join the corresponding bags on the cycle. Two semi-separators S, T are indicated. Notice that the union $S_3 \cup S_2$ forms a minimal separator of G. But it is not always the case. For example, $S_2 \cup S_3$ is not a minimal separator.

As we will see later, semi-separators are very useful in the analysis of folding. Another notion that we need to introduce for this purpose is the restriction of a clique cycle to a subset of cliques.

Definition 7.2.2. *Let G be a circular-arc graph with a clique cycle $(\mathcal{X}, C)$ and let $\mathcal{A}$ be a set of bags on the cycle, a subset of $\mathcal{X}$. The* restriction *of the clique cycle $(\mathcal{X}, C)$ to the set of cliques in $\mathcal{X}' = \mathcal{X} \setminus \mathcal{A}$ is obtained by removing each $X \in \mathcal{A}$ from the model, and making former neighbors of X adjacent. The vertices that no longer belong to any clique disappear from the model.*

Clearly, $(\mathcal{X}', C')$ - the restriction of $(\mathcal{X}, C)$ to $\mathcal{X}'$ is again a clique cycle of some circular-arc subgraph of G. In Figure 7.3 we give an example of this operation. The set of cliques $\mathcal{X}$ has been restricted to a subset $\mathcal{X}'$.

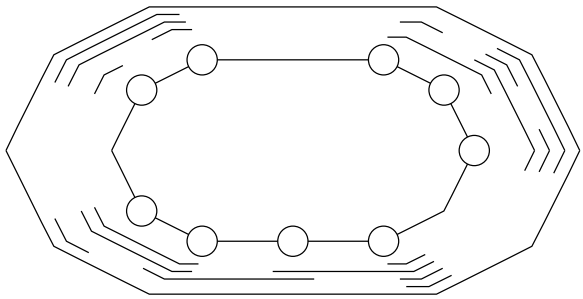


Figure 7.3: Restriction of the clique cycle.

7.3 Folding circular-arc graphs

Let $(\mathcal{X}, C)$ be the clique cycle of the circular-arc graph $G = (V, E)$. Consider a permutation $\mathcal{Q}$ of the set of bags $\mathcal{X}$. In the case of the circular-arc graphs, we study the permutation $\mathcal{Q}$ with respect to the circular ordering of the cliques on the cycle of the decomposition. Therefore it is more convenient to think of a folding as a *triple* $(\mathcal{X}, C, \mathcal{Q})$.

Remark 7.3.1. *Let $(\mathcal{X}, C, \mathcal{Q})$ be a folding of the circular-arc graph G. Consider the clique path decomposition P produced by the algorithm `FillFolding`$(G, \mathcal{Q})$. Observe that each bag of P is the union of the clique $Q \in \mathcal{Q}$ which corresponds to the bag at the initialization step, and some semi-separators of type $Q' \cap Q''$, where Q', Q'' are two cliques consecutive on the cycle, but separated by Q in $\mathcal{Q}$. We say that the clique Q and the semi-separators have been* merged *by the folding.*

In the upper part of Figure 7.4, there is a sample circular-arc graph G represented by a clique cycle. The lower part presents the clique path of $H =$ `FillFolding`$(G, \mathcal{Q})$, produced by the algorithm `FillFolding` on the sequence $\mathcal{Q} = (Q_1, \ldots, Q_5)$. Each clique Q'_i of H is the union of Q_i and the corresponding semi-separator S_i - except for the first and the last clique in $\mathcal{Q}$, which stay unchanged.

A folding $(\mathcal{X}, C, \mathcal{Q})$ naturally defines an upper part and a lower part of the cycle $(\mathcal{X}, C)$. Let Q_L, Q_R be the leftmost and rightmost element of the permutation $\mathcal{Q}$. Let $\mathcal{X}^{down}$ ($\mathcal{X}^{down}$) denote the cliques counterclockwise (clockwise) between Q_L and Q_R on the cycle. Let $\mathcal{Q}^{down} = (Q_L = Q_{l_1}, Q_{l_2}, \ldots, Q_{l_r} = Q_R)$ denote the restriction of $\mathcal{Q}$ to $\mathcal{X}^{down}$. Similarly let $\mathcal{Q}^{up} = (Q_L = Q_{u_1}, Q_{u_2}, \ldots, Q_{u_t} = Q_R)$ denote the restriction of $\mathcal{Q}$ to $\mathcal{X}^{up}$.

Definition 7.3.2. *Given a clique cycle decomposition $(\mathcal{X}, C)$ of G and a permutation $\mathcal{Q}$ of $\mathcal{X}$, we say that a clique $X \in \mathcal{X}$ is a* pivot *of the folding $(\mathcal{X}, C, \mathcal{Q})$ if its neighbors on the cycle appear on*

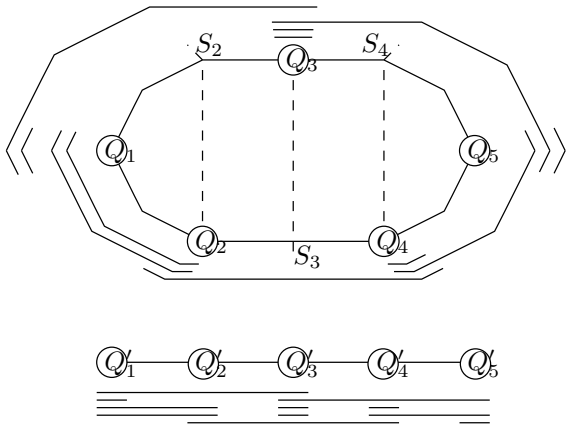


Figure 7.4: `FillFolding`.

the same side of X in Q. We extend this definition to any subset $\mathcal{X}'$ of $\mathcal{X}$: $X \in \mathcal{X}'$ is a pivot w.r.t. $\mathcal{X}'$ if $X_L, X_R \in \mathcal{X}'$, its closest neighbors on the cycle among the elements of $\mathcal{X}'$, are on the same side of X in $\mathcal{Q}$.

Clearly, any folding $(\mathcal{X}, C, \mathcal{Q})$ has at least two pivots, namely the first and the last element of the permutation $\mathcal{Q}$.

Definition 7.3.3. *Let $(\mathcal{X}, C)$ be a clique cycle decomposition of $G = (V, E)$. A permutation $\mathcal{Q}$ of a subset $\mathcal{X}' \subseteq \mathcal{X}$, is* k-monotone *if it contains exactly k pivots. The* monotonicity *of $\mathcal{Q}$ is the minimum k such that $\mathcal{Q}$ is k-monotone. The monotonicity of a folding $(\mathcal{X}, C, \mathcal{Q})$ is the monotonicity of $\mathcal{Q}$.*

The main combinatorial result of the chapter consists in proving that there exists a 2-monotone folding $(\mathcal{X}, C, \mathcal{Q})$ such that $H = \texttt{FillFolding}(G, \mathcal{Q})$ is an interval completion of G satisfying $\text{pwd}(H) = \text{pwd}(G)$. Therefore, the optimum interval completion for the pathwidth problem can be found among the completions defined by 2-monotone foldings. In a two-monotone folding, the only pivots are the first and last element of $\mathcal{Q}$. Moreover, $\mathcal{Q}^{up}$ ($\mathcal{Q}^{down}$) is clockwise (counterclockwise) consecutive on the cycle $(\mathcal{X}, C)$.

The following lemma is straightforward (see also Remark 7.3.1).

Lemma 7.3.4. *Let $(\mathcal{X}, C, \mathcal{Q})$ be a 2-monotone folding and let P be the clique path decomposition produced by `FillFolding`$(G, \mathcal{Q})$. Every bag of P is the union of a clique $Q \in \mathcal{Q}$ and of a unique semi-separator corresponding to the edge $\{Q', Q''\}$ of the cycle, such that Q separates Q' and Q'' in the permutation $\mathcal{Q}$.*

We aim towards proving that there always exists a 2-monotone folding of G that defines an interval completion of pathwidth equal the pathwidth of G. But first, we need to show that every 4-monotone folding can be transformed into a 2-monotone folding without augmenting the pathwidth of the corresponding completion. For this purpose, we define the anomaly of a 4-monotone folding, corresponding to the part that has to be rearranged in order to obtain a 2-monotone folding.

Definition 7.3.5. *Let $(\mathcal{X}, C)$ be a clique cycle decomposition of $G = (V, E)$ and let $(\mathcal{X}, C, \mathcal{Q})$ be a 4-monotone folding. Let Q_L, Q_R be the end cliques (pivots) of $\mathcal{Q}$. Let B_1, P be the other pivots,*

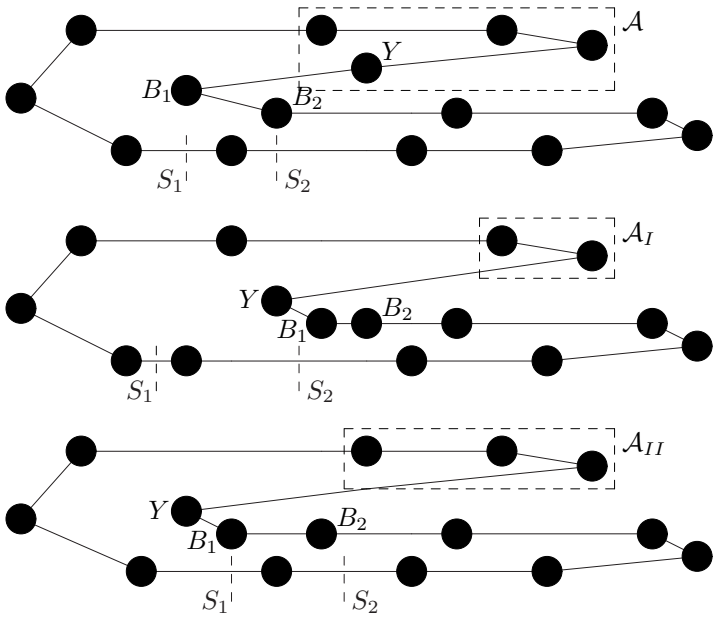


Figure 7.5: Reduction of $\mathcal{A}$ (top) to $\mathcal{A}'$: one way (middle) or the other (bottom).

ordered as in $\mathcal{Q}$. Assume w.l.o.g. that B_1, P belong to $\mathcal{Q}^{up}$. The consecutive part of the cycle that appears counterclockwise, starting right after B_1, passing through P, continuing as long as it stays after B_1 in $\mathcal{Q}$ is called the anomaly *(see the top part of Figure 7.5).*

Notice that for a 4-monotone folding $\mathcal{Q}$, the restriction of $\mathcal{Q}$ to $\mathcal{X} \setminus \mathcal{A}$ is 2-monotone.

One of our main tools (Theorem 7.3.7) shows that if $(\mathcal{X}, C, \mathcal{Q})$ is a 4-monotone folding which defines $H = \texttt{FillFolding}(G, \mathcal{Q})$, then there exists a 2-monotone folding $(\mathcal{X}, C, \mathcal{Q}')$ defining an interval graph $H' = \texttt{FillFolding}(G, \mathcal{Q}')$ of pathwidth smaller or equal the pathwidth of H. Here is an informal sketch of the main idea. Consider an anomaly $\mathcal{A}$ of the 4-monotone folding $(\mathcal{X}, C, \mathcal{Q})$. Suppose that the anomaly is in the upper part of the cycle, as on Figure 7.5.

Recall that when we restrict $(\mathcal{X}, C)$ to the cliques that are not in the anomaly (just remove every $X \in \mathcal{A}$, making the former neighbors of X adjacent), then we obtain a clique cycle of $\overline{G} = G[\bigcup(X \setminus \mathcal{A})]$, an induced subgraph of G. Moreover, we obtain $(\overline{\mathcal{X}}, \overline{C}, \overline{\mathcal{Q}})$, a folding of $\overline{G}$, where $\overline{\mathcal{Q}}$ is the restriction of sequence $\mathcal{Q}$ to the cliques not in the anomaly $\overline{\mathcal{X}} = \mathcal{X} \setminus \mathcal{A}$. It defines an interval completion $\overline{H} = \texttt{FillFolding}(\overline{G}, \overline{\mathcal{Q}})$.

Definition 7.3.6. *Let $(\mathcal{X}, C)$ be a clique cycle decomposition of $G = (V, E)$ and let $(\mathcal{X}, C, \mathcal{Q})$ be a 4-monotone folding with the anomaly $\mathcal{A}$. The* $\mathcal{A}$-width *of $(\mathcal{X}, C, \mathcal{Q})$ is the pathwidth of $\overline{H} = \texttt{FillFolding}(\overline{G}, \overline{\mathcal{Q}})$, where $\overline{G}$ is the circular-arc graph defined by the clique cycle $(\mathcal{X}, C)$ restricted to $\overline{\mathcal{X}} = \mathcal{X} \setminus \mathcal{A}$ and $\overline{\mathcal{Q}}$ is the sequence $\mathcal{Q}$ restricted to $\overline{\mathcal{X}}$.*

One step of the procedure is to slightly modify the folding $(\mathcal{X}, C, \mathcal{Q})$ to obtain $(\mathcal{X}, C, \mathcal{Q}')$ with a strictly smaller anomaly $\mathcal{A}'$, ensuring that the $\mathcal{A}'$-width of $(\mathcal{X}, C, \mathcal{Q}')$ is not bigger than the pathwidth of H. Continue until the anomaly is empty. Eventually, this yields a folding $(\mathcal{X}, C, \mathcal{Q}'')$ which is 2-monotone. Its anomaly $\mathcal{A}''$ being empty, the $\mathcal{A}''$-width of this folding is equal to the pathwidth of $H'' = \texttt{FillFolding}(G, \mathcal{Q}'')$, and it is not bigger than the pathwidth of H.

Let us give in more detail the construction of $(\mathcal{X}, C, \mathcal{Q}')$ based on $(\mathcal{X}, C, \mathcal{Q})$. Consider the pivots of $(\mathcal{X}, C, \mathcal{Q})$ that are not end-cliques of $\mathcal{Q}$. Let P be the one that belongs to the anomaly $\mathcal{A}$, and let B_1 be the other one. Let B_{k+1} be the neighbor of B_1 on the cycle that belongs to the anomaly. Let $B_2, \ldots, B_k$ be the cliques, which do not belong to the anomaly, that follow B_1 clockwise on the cycle and appear before B_{k+1} in $\mathcal{Q}$. Let $S_1, \ldots, S_{k+1}$ be the semi-separators on the lower part of the cycle, such that S_i is merged with the corresponding B_i in $\texttt{FillFolding}(G, \mathcal{Q})$. In this setting, we choose a semi-separator S_l and permute $\mathcal{Q}$ in order to put all $B_{k+1}, B_1, \ldots, B_{l-1}$ (in this order) between Q and Q', where Q, Q' are the consecutive cliques in the lower part of the cycle such that $S_l = Q \cap Q'$. We choose S_l such that the new folding $(\mathcal{X}, C, \mathcal{Q}')$ has the desired property. We say that in such a situation we *put* $B_{k+1}, B_1, \ldots, B_{l-1}$ *on* the semi-separator S_l. This construction is illustrated in Figure 7.5. Informally, the bags $B_{k+1}, B_1, \ldots, B_{l-1}$ slide (without jumping) along the cycle, in the clockwise sense, and they stop above the edge of the cycle corresponding to S_l.

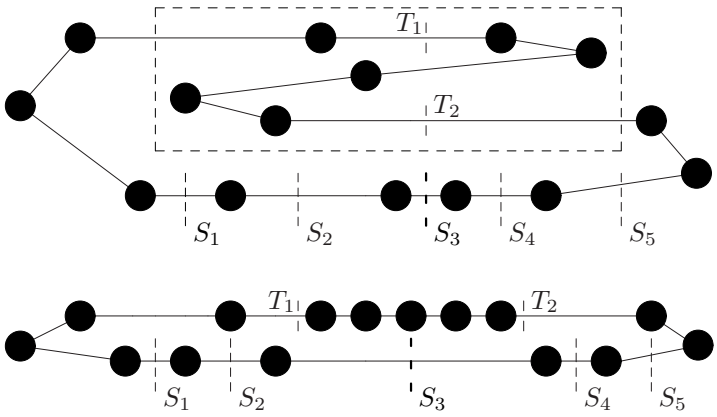


Figure 7.6: From 4-monotone to 2-monotone foldings.

Theorem 7.3.7. *Let $(\mathcal{X}, C)$ be a clique cycle decomposition of $G = (V, E)$. Let $(\mathcal{X}, C, \mathcal{Q})$ be a 4-monotone folding and $H = \texttt{FillFolding}(G, \mathcal{Q})$. Then there is a 2-monotone folding $(\mathcal{X}, C, \mathcal{Q}')$ such that the pathwidth of $H' = \texttt{FillFolding}(G, \mathcal{Q}')$ is not bigger than the pathwidth of H. Moreover, we can assume that $(\mathcal{X}, C, \mathcal{Q}')$ is such that $\mathcal{X}'^{up} = \mathcal{X}^{up}$ and $\mathcal{X}'^{down} = \mathcal{X}^{down}$.*

Proof. Let $(\mathcal{X}, C, \mathcal{Q}')$ be a ≤ 4-monotone folding with the anomaly $\mathcal{A}'$, such that $\mathcal{X}'^{up} = \mathcal{X}^{up}$ and $\mathcal{X}'^{down} = \mathcal{X}^{down}$, which satisfies the following Properties:

1. $\mathcal{A}' \subseteq \mathcal{A}$;
2. for any clique Q of $\mathcal{A}'$, the semi-separator S of the lower part of the cycle that is merged to Q in H' is the same as in H;
3. the $\mathcal{A}'$-width of $(\mathcal{X}, C\mathcal{Q}')$ is not bigger than the pathwidth of H,
4. the anomaly $\mathcal{A}'$ is inclusion minimal among all such foldings.

Let us show that $\mathcal{A}'$ is in fact empty, thus the pathwidth of H' is not bigger than the pathwidth of H. Suppose $\mathcal{A}'$ is not empty.

We use the notations introduced in the informal description above. Let us use Y and S_Y as shorthands for B_{k+1} and S_{k+1}. By Property 2 and Remark 7.3.1, we have:

$$|Y \cup S_Y| \leq \text{pwd}(H) + 1. \tag{7.1}$$

The semi-separators S_i, $1 \leq i \leq k+1$, can be partitioned as follows:

$$\begin{gathered} S_i = N_i^j \cup B_i^j \cup Y_i^j \cup I_i^j \text{ , for any } 1 \leq j \leq k \text{ , where} \\ N_i^j = S_i \setminus (B_j \cup Y), B_i^j = S_i \cap B_j \setminus Y, Y_i^j = S_i \cap Y \setminus B_j, I_i^j = S_i \cap B_j \cap Y. \end{gathered} \tag{7.2}$$

Claim 8. *For any $1 \leq i \leq k$, $1 \leq p \leq q \leq k+1$, one of the following holds:*

$$|B_i \cup S_p| \geq |B_i \cup S_q| \text{ or } |Y \cup S_q| \geq |Y \cup S_p| \tag{7.3}$$

Proof. Suppose it is not true. By Equation 7.2, we have:

$$|B_i \cup N_p^i \cup Y_p^i| < |B_i \cup N_q^i \cup Y_q^i| \text{ and } |Y \cup N_q^i \cup B_q^i| < |Y \cup N_p^i \cup B_p^i|,$$

which yields a contradiction: $|N_p^i| < |N_q^i|$ and $|N_q^i| < |N_p^i|$, since for any j, p, q, $1 \leq j \leq k$, $1 \leq p \leq q \leq k+1$ there is $Y_q^j \subseteq Y_p^j$ and $B_p^j \subseteq B_q^j$, by properties of the clique cycle.

□

Claim 9. *Let l be the biggest integer such that $|Y \cup S_Y| < |Y \cup S_i|$, for $1 \leq i \leq l-1$. Then*

$$|Y \cup S_Y| \geq |Y \cup S_l|, \tag{7.4}$$

$$|B_i \cup S_i| \geq |B_i \cup S_l|, \text{ for any } 1 \leq i \leq l, \tag{7.5}$$

Proof. The first equation is clear from the construction. Since $|Y \cup S_l| \leq |Y \cup S_Y|$ and $|Y \cup S_Y| < |Y \cup S_i|$, for any $1 \leq i \leq l-1$, there is $|Y \cup S_l| < |Y \cup S_i|$, for any $1 \leq i \leq l-1$. Now, by Equation 7.3, for any $1 \leq i \leq l-1$, we get $|B_i \cup S_i| \geq |B_i \cup S_l|$, for any $1 \leq i \leq l-1$.

□

Therefore, by putting Y and all B_i, for $1 \leq i \leq l-1$, on S_l, we create a new folding $(\mathcal{X}, C, \mathcal{Q}'')$ with a strictly smaller anomaly $\mathcal{A}''$. Indeed, there is $Y \in \mathcal{A}' \setminus \mathcal{A}''$. Notice there may be other cliques in $\mathcal{A}' \setminus \mathcal{A}''$ as well.

Let us check that the $\mathcal{A}''$-width of the folding $(\mathcal{X}, C, \mathcal{Q}'')$ is at most $\text{pwd}(H)$. For each clique X of $\mathcal{Q}'' \setminus \mathcal{A}''$, let S_X be the (unique) semi-separator of the lower part of the cycle to which X is merged in `FillFolding`(G, Q''). The $\mathcal{A}''$-width of $(\mathcal{X}, C, \mathcal{Q}'')$ is the maximum, over all cliques X, of $|X \cup S_X| - 1$. We check that $|X \cup S_X| - 1$ is at most the $\text{pwd}(H)$, for every clique X. If X is also in $\mathcal{Q}' \setminus \mathcal{A}'$, this quantity is upper bounded by the $\mathcal{A}'$-width of $(\mathcal{X}, C, \mathcal{Q}')$ and the conclusion follows by Property 3. If $X = Y$, then $S_X = S_l$ and the conclusion follows from Equations 7.4 and 7.1. If X is one of the B_i's, $1 \leq i \leq l-1$, again $S_X = S_l$ and the conclusion follows from Equation 7.5 and Property 3. Finally, if X is one of the cliques of $\mathcal{A}' \setminus \mathcal{A}''$, different from Y, the conclusion follows from Property 2.

The new folding $(\mathcal{X}, C, \mathcal{Q}'')$ also respects Property 2, since in the permutation $\mathcal{Q}''$ the cliques of $\mathcal{A}''$ have the same position w.r.t. the lower part of the cycle as before.

The construction of $\mathcal{Q}''$ contradicts Property 4 of $\mathcal{Q}'$. So $\mathcal{A}'$ must be empty.

□

Now that we have transformed a 4-monotone folding into a 2-monotone one, we use an inductive argument to prove that any folding can be transformed into a 2-monotone one without increasing the pathwidth of the corresponding completion.

Theorem 7.3.8. *Let $(\mathcal{X}, C)$ be a clique cycle decomposition of $G = (V, E)$. There is a folding $(\mathcal{X}, C, \mathcal{Q})$, with $\mathcal{Q}$ being a permutation of $\mathcal{X}$, such that $\mathcal{Q}$ is 2-monotone and $H = \texttt{FillFolding}(G, \mathcal{Q})$ is an interval completion of G of pathwidth equal the pathwidth of G.*

Proof. Let $(\mathcal{X}, C, \mathcal{Q})$ be a folding of minimum monotonicity such that the pathwidth of $H = \texttt{FillFolding}(G, \mathcal{Q})$ is not bigger that the pathwidth of G. We will prove that it is 2-monotone.

Suppose it is not. Assume w.l.o.g. that $\mathcal{X}^{up}$ contains some pivots other than Q_L, Q_R. Let B_1 be the leftmost pivot in $\mathcal{Q}^{up}$. Let P be the rightmost in $\mathcal{Q}^{up}$ among the pivots which are between Q_L and B_1 clockwise on the cycle $(\mathcal{X}, C)$. Let $\mathcal{Q}_L^{up}$ denote the subsequence of $\mathcal{Q}^{up}$ induced by cliques clockwise between Q_L and P (included) on the cycle. Let $\mathcal{Q}_C^{up}$ denote the subsequence of $\mathcal{Q}^{up}$ induced by cliques between P and B_1 (included), and $\mathcal{Q}_R^{up}$ denote the subsequence of $\mathcal{Q}^{up}$ induced by cliques between B_1 and Q_R (included). Let G_L^{up} be the graph defined by the folding $\mathcal{Q}_L^{up}$, restricted to the corresponding set of bags: $G_L^{up} = \texttt{FillFolding}(G[\bigcup \mathcal{Q}_L^{up}], \mathcal{Q}_L^{up})$. We denote by P_L^{up} the clique path decomposition produced by the folding algorithm. Similarly, we define G_C^{up}, G_R^{up} and G^{down}, with the corresponding clique path decompositions. Let $\tilde{G}$ be the union of these four graphs. Note that $\tilde{G}$ is a circular-arc graph. A clique cycle decomposition $(\tilde{\mathcal{X}}, \tilde{C})$ of $\tilde{G}$ is obtained by gluing into a cycle the paths P_L^{up}, the reverse of P_C^{up}, P_R^{up}, and to the reverse of P^{down}. The gluing is performed by identifying the bags P, then B_1, Q_R and finally Q_L.

Moreover, this procedure yields a folding $(\tilde{\mathcal{X}}, \tilde{C}, \tilde{\mathcal{Q}})$ of $\tilde{G}$. The bags of $\tilde{\mathcal{X}}$ are in one-to-one correspondence to the bags of $\mathcal{X}$, so the permutation $\mathcal{Q}$ of $\mathcal{X}$ is translated into a permutation $\tilde{\mathcal{Q}}$ of $\tilde{\mathcal{X}}$. Notice that $(\tilde{\mathcal{X}}, \tilde{C}, \tilde{\mathcal{Q}})$ is a 4-monotone folding, since Q_L, B_1, P, Q_R are the only pivots left. Also $\texttt{FillFolding}(\tilde{G}, \tilde{\mathcal{Q}}) = H = \texttt{FillFolding}(G, \mathcal{Q})$.

By Theorem 7.3.7 on $(\tilde{\mathcal{X}}, \tilde{C}, \tilde{\mathcal{Q}})$ there is a 2-folding $(\tilde{\mathcal{X}}, \tilde{C}, \tilde{\mathcal{Q}}')$ such that the pathwidth of $H' = \texttt{FillFolding}(\tilde{G}, \tilde{\mathcal{Q}}')$ is not bigger than the pathwidth of H, thus not bigger than the pathwidth of G.

Since $\tilde{\mathcal{Q}}'$ is 2-monotone and $\tilde{\mathcal{X}}'^{up} = \tilde{\mathcal{X}}^{up}$, the only pivots of $\tilde{\mathcal{Q}}'$ are Q_L and Q_R. Notice that there is : $\tilde{\mathcal{Q}}'^{up}$ equals P_L^{up} glued to the reverse of P_C^{up} glued to P_R^{up}, and $\tilde{\mathcal{Q}}'^{down}$ equals P^{down}.

Because of the one-to-one correspondence between the elements of $\mathcal{X}$ and $\tilde{\mathcal{X}}$, we construct the folding $(\mathcal{X}, C, \mathcal{Q}')$ directly from $(\tilde{\mathcal{X}}, \tilde{C}, \tilde{\mathcal{Q}}')$, by just replacing the elements of $\tilde{\mathcal{X}}$ with the corresponding elements of $\mathcal{X}$. Clearly, B_1 and P are not pivots of $\mathcal{Q}'$, whereas all the other pivots of $\mathcal{Q}'$ are also pivots of $\mathcal{Q}$. Moreover, it is easy to verify that $\texttt{FillFolding}(G, \mathcal{Q}') = \texttt{FillFolding}(\tilde{G}, \tilde{\mathcal{Q}}')$. Therefore, $(\mathcal{X}, C, \mathcal{Q}')$ is a folding of strictly smaller monotonicity than $(\mathcal{X}, C, \mathcal{Q}')$, which also defines a completion of pathwidth not bigger that the pathwidth of G. A contradiction.

□

7.4 The algorithm

Based on Theorem 7.3.8, the algorithm for computing the pathwidth of circular-arc graphs is very similar to the algorithm computing the minimum fill-in for the same class of graphs [86].

Consider a clique cycle $CC_{G,M} = (\mathcal{X}, C)$ of the input graph G. Subdivide each edge of the cycle by adding a new bag containing the semi-separator corresponding to the edge. We obtain a *clique-semi-separator* cycle alternating original clique bags and semi-separator bags. We should

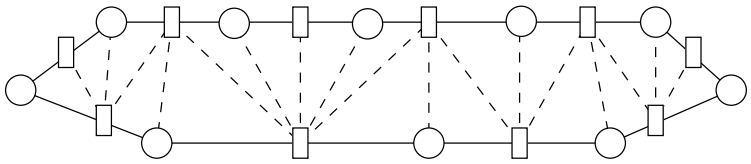


Figure 7.7: Planar triangulation corresponding to a 2-folding.

also see this cycle as a (regular) *polygon of scanpoints* $\mathcal{P}_G$, following the terminology of [86]; the scanpoints are the clique and semi-separator bags of the cycle. Therefore we associate to each scanpoint s the set of vertices $V(s)$, corresponding to the clique or semi-separator represented by the scanpoint. For each triangle T formed by three scanpoints s_1, s_2, s_3 , define the width $w(T)$ of the triangle as the cardinality of the union $V(s_1) \cup V(s_2) \cup V(s_3)$.

Definition 7.4.1. *A* linear (planar) triangulation *LP of the polygon of scanpoints $\mathcal{P}_G$ is a planar triangulation such that every triangle contains at most two diagonals. The width $w(LT)$ of the linear triangulation is the maximum width of its faces (triangles).*

In Figure 7.7, there is an example of a 2-folding. The cliques on the clique-semi-separator cycle are marked with circles and the semi-separators are the rectangles. The linear planar triangulation corresponding to this folding is drown with dashed lines.

First, we show that the pathwidth of G equals the minimum width of a linear planar triangulation, minus one. Eventually we give an algorithm computing a linear triangulation of minimum width.

Lemma 7.4.2. *Let LT be a linear planar triangulation of the polygon of scanpoints $\mathcal{P}_G$. There is a path decomposition of G, of width $w(LT) - 1$.*

Proof. Consider a graph whose vertex set is the set of triangles of LT, and such that two vertices are adjacent if and only if the corresponding triangles share a diagonal. This graph is called the inner dual of LT. Since LT is a linear triangulation, the maximum degree of the inner dual is 2. Clearly this graph is connected and contains no cycle, so it is a path P. The path decomposition of G is constructed using the path P. For each node T of P, let s_1, s_2, s_3 be the three corresponding scanpoints. The bag associated to T is $V(s_1) \cup V(s_2) \cup V(s_3)$. It remains to show that P and its bags form indeed a path decomposition of G. By contradiction, suppose that a vertex x of G appears in the bags corresponding to some nodes T, T' of P but not in the bag of T'', on the T, T'-subpath of P. Let D be an edge of T'', corresponding to a diagonal of $\mathcal{P}_G$. Since x is in the bags T and T', the circular arc corresponding to x intersects both sides of the cycle, with respect to the diagonal D. Hence at least one end-point of the diagonal D is on the circular-arc x – a contradiction. Clearly, the width of this path decomposition is $w(LT) - 1$.

□

Lemma 7.4.3. *Let $(G, \mathcal{Q})$ be a 2-monotone folding of G. There exists a linear planar triangulation $LT(\mathcal{Q})$ of $\mathcal{P}_G$ such that the width of $LT(\mathcal{Q})$ is equal to the pathwidth of* `FillFolding`*$(G, \mathcal{Q})$, plus one.*

Proof. The folding being 2-monotone, both $\mathcal{Q}^{up}$ and $\mathcal{Q}^{down}$ are increasing subsequences of Q. For every couple (Q', Q'') of consecutive cliques of $\mathcal{Q}^{down}$, let $(Q_{u_i}, Q_{u_{i+1}}, \ldots, Q_{u_j})$ be the subsequence of $\mathcal{Q}^{up}$ appearing strictly between Q' and Q'' in Q. Add, in $\mathcal{P}_G$, a diagonal between

- The scanpoint s corresponding to the semi-separator of the edge $\{Q', Q''\}$ and every scan-point corresponding to a clique of $(Q_{u_i}, Q_{u_{i+1}}, \ldots, Q_{u_j})$.
- The scan-point s and every scanpoint on some edge of the cycle incident to a clique of $(Q_{u_i}, Q_{u_{i+1}}, \ldots, Q_{u_j})$, including the edge preceding Q_{u_i} and the edge after Q_{u_j}.

The symmetric operation is performed by permuting the role of $\mathcal{Q}^{up}$ and $\mathcal{Q}^{down}$.

It is not hard to check that by adding this set of diagonals we obtain indeed a linear triangulation $LT(\mathcal{Q})$ of the polygon of scanpoints. Each triangle T of $LT(\mathcal{Q})$ has exactly two semi-separator scanpoints and a clique-scanpoint. More precisely, for each triangle T the clique scan-point Q_T is incident to one of the semi-separator scanpoints. The other scanpoint is either incident to Q_T (if Q_T is the first or last clique of $\mathcal{Q}$) or it corresponds to an edge $\{Q_a, Q_b\}$ of the clique cycle, such that Q_T is between Q_a and Q_b in Q. Note that $Q_T \cup (Q_a \cap Q_b)$ is also a clique of the graph $H = \texttt{FillFolding}(G, Q)$, by the construction of the graph H. It follows that $w(LT(\mathcal{Q})) \leq \text{pwd}(H) + 1$. Conversely, consider any three cliques Q_a, Q_b, Q_T such that Q_T is between Q_a and Q_b in Q and $\{Q_a, Q_b\}$ is an edge of the clique cycle. Then $LT(\mathcal{Q})$ contains a diagonal D such that one of its end-points corresponds to Q_T and the other is the semi-separator scanpoint corresponding to the edge $\{Q_a, Q_b\}$. It follows that $Q_T \cup (Q_a \cap Q_b) \leq w(LT(\mathcal{Q}))$. By Lemma 7.3.4, every bag of the path decomposition of H, obtained by the folding algorithm, is of type $Q_T \cup (Q_a \cap Q_b)$ – and the conclusion follows. □

Theorem 7.4.4. *For any circular-arc graph G, its pathwidth is the minimum, over all linear planar triangulations LT of $\mathcal{P}_G$, of $w(LT) - 1$.*

Proof. By Lemma 7.4.2, we have that $\text{pwd}(G) \leq \min w(LT) - 1$. By Theorem 7.3.8, there is a 2-monotone folding (G, Q) of G such that $\text{pwd}(\texttt{FillFolding}(G, Q)) = \text{pwd}(G)$. The conclusion follows by Lemma 7.4.3.

□

The algorithm for computing a linear triangulation of $\mathcal{P}_G$ of minimum width is very similar to the one of [86].

Theorem 7.4.5. *The pathwidth of circular-arc graphs can be computed in $\mathcal{O}(n^2)$ time.*

Proof. It remains to give an algorithm computing a linear planar triangulation LT of $\mathcal{P}_G$, of minimum width. The algorithm is practically the same as in [86], except that in their case the planar triangulation is not necessarily linear and that the width function is slightly different. Therefore we only give a short description of the algorithm.

Observe that, for computing linear planar triangulations, we only need $\mathcal{O}(n^2)$ triangles. Indeed, for each triangle corresponding to a face, two endpoints are consecutive on the polygon. Assume first that we are given the widths of all the triangles of this type. Let $s_0, s_1, \cdots, s_{p-1}$ be the scanpoints of $\mathcal{P}_G$, ordered by the counter-clockwise orientation of the polygon. Let $w(i, j)$ the optimum width, over all linear triangulations $LT(i, j)$ of the polygon formed by the points $s_i, s_{i+1} \ldots, s_j$ (indices are considered modulo p). We point out that, if (s_i, s_j) is a diagonal of $\mathcal{P}_G$, then it is also considered as a diagonal of the restricted polygon, meaning that $LT(i, j)$ is not allowed to have a face with (s_i, s_j) and two other diagonals as edges.

If $j = i + 2$, then $w(i, j)$ is the width of the triangle (s_i, s_{i+1}, s_j). If $j > i + 2$, for using the diagonal (s_i, s_j) we must also use one of the diagonals (s_{i+1}, s_j) or (s_i, s_{j-1}). Therefore:

$$w(i, j) = \min\left(\max(w(s_i, s_{i+1}, s_j), w(i + 1, j)), \max(w(s_i, s_{j-1}, s_j), w(i, j - 1))\right)$$

The optimum width of a linear triangulation is

$$w(\mathcal{P}_G) = \min_{0 \leq i,j < p, j \geq i+2} \max(w(i,j), w(j,i))$$

All these equations are easily transformed into an $\mathcal{O}(n^2)$ dynamic programming algorithm for computing the linear planar triangulation of optimum width. Within the same time bounds, we can equally obtain, like in Lemma 7.4.2, an optimum path decomposition of the input graph.

The widths of the triangles can be computed in $\mathcal{O}(n^3)$, so the pathwidth of G can be computed in $\mathcal{O}(n^3)$. Following the principles of [86], we can improve this running time by computing the widths of the needed triangles in $\mathcal{O}(n^2)$. Each triangle T of a linear planar triangulation is of the type (s_i, s_{i+1}, s_j), with two consecutive scanpoints. One of the two, say s_i, is a clique scanpoint. Hence the width of T is $|V(s_j) \cup V(s_i)|$. Thus we only need to compute, for every couple $0 \leq j < i < p$ the cardinality of $V(s_j) \cup V(s_i)$. During a preprocessing step we explore the clique-semi-separator cycle (in counter-clockwise order). For each scanpoint s_i, we distinguish two situations, depending whether s_i corresponds to a semi-separator or a clique. In the first case, s_i is a clique bag. We compute the arcs of $V(s_i) \setminus V(s_{i-1})$. Moreover, for each $j, 1 \leq j < p$ let $Add(i,j)$ be the number of circular-arcs x of $V(s_i) \setminus V(s_{i-1})$ such that the right-end point $r(x)$ is between i and $j-1$ according to the cyclic order. Using a bucket sort, these quantities can be computed in $O(n)$ for each i. In a similar way, if i is a semi-separator scanpoint we take into account the vertices of $V(s_{i-1}) \setminus V(s_i)$. Let $Sub(i,j)$ the number of arcs $x \in V(s_{i-1}) \setminus V(s_i)$ such that the left-end point $l(x)$ is between $j+1$ and i in the cyclic order. Fix a value j, $0 \leq j < p$. Consider each i, from $j+1$ to $j-1$ in cyclic order. The value $|V(s_j) \cup V(s_{j+1})|$ is computed directly. Now observe that if s_i is a clique scanpoint, we have $|V(s_j) \cup V(s_i)| = |V(s_j) \cup V(s_{i-1})| + Add(i,j)$. Indeed, the only arcs of $V(s_j) \cup (V(s_i) \setminus V(s_{i-1}))$ ar the arcs of $V(s_i) \setminus V(s_{i-1})$ whose right endpoint is strictly smaller that j in the cyclic order. Similarly, if s_i is a semi-separator scanpoint then $|V(s_j) \cup V(s_i)| = |V(s_j) \cup V(s_{i-1})| - Sub(i,j)$. Thus we need a constant time to compute $|V(s_j) \cup V(s_i)|$, and $\mathcal{O}(n^2)$ to compute the weights of the useful triangles. □

7.5 Conclusions

We give in this chapter an $\mathcal{O}(n^2)$ time algorithm computing the pathwidth and the corresponding path decomposition of a circular-arc graphs. This is the first polynomial time algorithm computing the pathwidth of graphs, for a class of graphs of unbounded treewidth. This indicates that the tool of folding that we use can be very helpful for understanding pathwidth.

One natural direction of pursuing this research consists in analyzing intersection models of other graph classes. The goal is to find a class for which we could restrict the set of foldings which contains solutions optimal to pathwidth or profile. For circular-arc graphs, we have found that it is enough to browse through the set of 2-monotone foldings of a given clique cycle model. Can we find similar characterizations for circle graphs, trapezoid graphs or some other class? Another interesting question is if it is possible to give an analogous characterization of a set of foldings of circular-arc graphs that contains solutions optimal for profile or some other problems of embedding into interval graphs?

Chapter 8

Extraction of Minimal Interval Completion

Contents

In Chapters 6 and 4 we presented two algorithms that, given an arbitrary graph $G = (V, E)$, compute a minimal interval completion $H = (V, E \cup F)$ of G. Both algorithms are able to compute some completions - different outputs depend on the degrees of freedom that each algorithm has. But in neither case the freedom is wide enough to enable all possible completions in the output. Moreover, there are completions that can be obtained by one algorithm and not the other, in both ways. Then again, there is another question that neither of the algorithms can help answering: given an interval completion $H' = (V, E \cup F')$ of H, is it minimal? What we give in this chapter is a polynomial algorithm answering this question. What is more, this algorithm has all the freedom needed to yield any minimal interval completion of G. These two aspects are very important for heuristics computing interval completions close to the original graph. The former permits to take the output of any heuristic and remove redundant edges, that is the ones that are not necessary for the completion to be an interval graph. The latter makes it possible to start with a complete graph on V and guide the process of minimalization, based on the edges we prefer not to have in the interval completion of G.

8.1 Folding interval graphs

As mentioned before, minimal triangulations have this property that $H = (V, E \cup F)$ is a minimal triangulation of $G = (V, E)$ if and only if it is impossible to remove an edge $e \in F$ from H in a way that $H' = H - e$ is a triangulation of G. Minimal interval completions may not have this property. The ones that violate it are particularly hard to track down and we distinguish them with a special name.

Definition 8.1.1. *We say that H is a* quasi-minimal interval completion *of G if no single fill edge can be removed from H without destroying interval graph property. Simple examples exist to show that quasi-minimal interval completions are not necessarily minimal.*

Assume that we are given an interval completion H of an arbitrary graph G. We want to find out whether H is a minimal interval completion. First we can start by trying to remove every single fill edge and test, in linear time, whether the remaining graph is an interval graph. After a number of steps (which is at most quadratic in the number of edges of H) we reach a quasi-minimal interval completion. Thus from now on, we assume that we are given a quasi-minimal interval completion H of G, and we want to decide whether it is minimal. If it is not minimal, we know that there is one that is minimal which is a strict subgraph of H, and before we finally find a minimal one, we might explore several strict interval subgraphs of H that are not minimal.

Let us give different names to these different interval completions of G. Let H_2 be the given quasi-minimal interval completion of G. If it is not minimal, let H_0 be a minimal interval completion of G that is a subgraph of H_2. Since we are only given G and H_2, and we do not know H_0, we will probably discover several intermediate graphs H_1, where H_1 is an interval completion of G that is a strict subgraph of H_2. Hence we have the following relations between these graphs: $E(G) \subset E(H_0) \subseteq E(H_1) \subset E(H_2)$. The first subset relation is proper because we can always check before start whether or not G is already an interval graph, in linear time. Even though we do not know H_0, we know that H_2 is a non-minimal and quasi-minimal interval completion of it. In this section, we analyze the relations between H_0 and H_2. For this purpose we use the tool of folding introduced in Subsection 7.1. We investigate the particular case of folding interval graphs. It has some interesting properties with respect to quasi-minimal interval completions.

Let us restate the definition and the algorithm of folding in terms of interval graphs.

Definition 8.1.2. *Let H be any interval graph, let $\mathcal{Q}$ be any permutation of the set of maximal cliques of H. We say that $(H, \mathcal{Q})$ is a* folding *of H by $\mathcal{Q}$.*

The graph defined by a folding is not necessarily a quasi-minimal interval completion of H_0. Nevertheless, we prove in Theorem 8.1.8 that every quasi-minimal interval completion of H_0 is defined by some folding. Recall that we already know, by Theorem 7.1.3, that this property holds for minimal interval completions.

The maximal cliques of H_0 play a crucial role in the analysis of the relationship between H_0 and H_2; in particular the ones that are contained in a unique maximal clique of H_2. The following definition is general, however we will apply it later on interval input graphs and their non-minimal interval completions.

Definition 8.1.3. *Let H be an interval completion of an arbitrary graph G. A maximal clique of G which is a subset of exactly one maximal clique of H is called a* core clique *of G with respect to H.*

It is a straight forward consequence of Lemma 2.2.20, that a maximal clique of G which is not a core clique must be contained in some minimal separator of H.

Observation 8.1.4. *Let H be an interval completion of an arbitrary graph G. Let K be a maximal clique of G which is not a core clique with respect to H. Then there is a minimal separator S of H with $K \subseteq S$.*

Before we prove that any quasi-minimal interval completion H_2 of H_0 can be obtained via Algorithm `FillFolding`, we need to introduce a few lemmas that describe the structure of quasi-minimal interval completions.

The first lemma is very useful for proving that an interval completion which does not satisfy our characterization is, in fact, not quasi-minimal.

Lemma 8.1.5. *Let H_2 be a (non-minimal) interval completion of an interval graph H_0. Let xy be a fill-edge such that only one maximal clique of H_2 contains both x and y. Then H_2 is not a quasi-minimal interval completion of H_0.*

Proof. Let $P_2 = (\Omega_1, \Omega_2, ..., \Omega_k)$ be a clique-path of H_2 and suppose that Ω_i is the only clique containing both x and y. Observe that $H_1 = H_2 - xy$ is also an interval graph. Indeed, replacing Ω_i in P_2 with Ω_x, Ω_y, where $\Omega_x = \Omega_i \setminus \{y\}$ and $\Omega_y = \Omega_i \setminus \{x\}$ yields a path-decomposition of H_1, from which we can obtain a clique-path of H_1 by simply removing redundant cliques. □

Second lemma shows that every maximal clique of a quasi-minimal interval completion contains a core clique. It will be of much help to control foldings.

Lemma 8.1.6. *Let H_2 be a quasi-minimal interval completion of an interval graph H_0. Let Ω be a maximal clique of H_2. Then there is a core clique K of H_0 contained in Ω.*

Proof. Let $P_2 = (\Omega_1, \Omega_2, ..., \Omega_k)$ be a clique-path of H_2, and let $\Omega = \Omega_i$ for some $i \in \{1, ..., k\}$. Since Ω is a maximal clique, it contains a vertex x such that $x \notin \Omega_{i-1}$ and a vertex y such that $y \notin \Omega_{i+1}$. If $x = y$ then Ω is the unique maximal clique of H_2 containing x. In this case, it is sufficient to take a maximal clique K of H_0 containing x, and K is a core clique with respect to H_2. Consider the case $x \neq y$. If $xy \notin E(H_0)$, then by Lemma 8.1.5, H_2 is not quasi-minimal, and we have a contradiction to the premise of the lemma. Thus xy is an edge of H_0 contained only in Ω. Any maximal clique K of H_0 with $x, y \in K$ is contained only in Ω, and is thus a core clique. □

The last lemma tells that it is enough to analyze the positions of core cliques in some clique path of a quasi-minimal interval completion to know all its structure.

Lemma 8.1.7. *Let H_2 be a quasi-minimal interval completion of an interval graph H_0, and let P_2 be a clique-path of H_2. Let x be a vertex, and let P_x be the subpath of P_2 induced by the maximal cliques containing x. Then, each maximal clique of H_2 that is a leaf in P_x, contains a core clique that contains x.*

Proof. If x is simplicial in H_2 then P_x consists of a single maximal clique Ω. Thus any maximal clique K of H_0 with $x \in K$ is a core clique contained in Ω. Assume that P_x is not a singleton, and let Ω be a leaf clique of P_x. Let y be a vertex in Ω that is not contained in the clique next to Ω in P_x. Observe that $xy \in E(H_0)$, otherwise Lemma 8.1.5 would yield a contradiction. So we have an edge of H_0 contained only in Ω. Thus, there is a maximal clique of H_0 contained only in Ω. □

We are now ready to give the main result of this section. For the proof of the theorem, we need the following notation. Given any path-decomposition P and a vertex x of an arbitrary graph G, we denote by $L(x, P)$ and $R(x, P)$, respectively, the leftmost and rightmost bags of P containing x.

Theorem 8.1.8. *Let H_2 be a quasi-minimal interval completion of an interval graph H_0. Then there exists a folding $(H_0, \mathcal{Q})$ of H_0 such that $H_2 =$ `FillFolding`$(H_0, \mathcal{Q})$.*

Proof. Let $\mathcal{K}_0 = \{1 \leq i \leq p \mid K_i\}$ and $\mathcal{K}_2 = \{1 \leq i \leq k \mid \Omega_i\}$ denote the sets of maximal cliques of H_0 and H_2, respectively. Let $P_2 = (\Omega_1, \ldots, \Omega_k)$ be a clique-path of H_2. It defines a linear order on the set $\mathcal{K}_2$. In a natural way, P_2 defines a linear pre-order on $\mathcal{K}_0$ by

$$K_a \leq K_b \text{ if } K_a \subseteq \Omega_i \text{ and } K_b \subseteq \Omega_j, \text{ where } i \leq j \leq k \text{ and } a, b \leq p.$$

Transform it into a linear order (sequence) $\mathcal{Q}$ as follows. First, let $\mathcal{Q}$ be a linear ordering of the core cliques according to the relative ordering in P_2 of the unique cliques that contain them (i.e., by using any linear extension of the pre-order above). Notice that for any maximal clique K of H_0 which is not a core clique, by Lemma 8.1.7, there is a core-clique K' in $\mathcal{Q}$, such that for every vertex $x \in K$, there are two core-cliques $K_l(x), K_r(x)$ containing x, such that $K_l(x) \leq K'$ and $K_r(x) > K'$ in $\mathcal{Q}$. Indeed, by Observation 8.1.4 and Lemma 2.2.20, there are two maximal cliques Ω', Ω'' consecutive on P_2, such that $K \subseteq S = \Omega' \cap \Omega''$. Then for every vertex $x \in K$, by Lemma 8.1.7, there is some core clique $K_l(x)$ ($K_r(x)$) contained in a bag to the left (right, respectively) of S in P_2. Insert K as the immediate successor of K' in $\mathcal{Q}$. This operation does not change the values of $L(x, \mathcal{Q})$ and $R(x, \mathcal{Q})$.

Let us define $H_2' = \texttt{FillFolding}(H_0, \mathcal{Q})$, and prove that $H_2' = H_2$. Notice that $xy \in E(H_2)$ if and only if the intervals defined by endpoints $L(x, P_2), R(x, P_2)$ and $L(y, P_2), R(y, P_2)$ intersect in P_2. Clearly, the same holds for H_2' and any clique-path P_2' of H_2'. By construction, $E(H_0) \subseteq E(H_2') \subseteq E(H_2)$. Let us prove that all fill-edges of H_2 are also present in H_2'. Suppose it is not the case. Then there is a fill-edge $xy \in E(H_2) \setminus E(H_2')$. Thus the intervals corresponding to x,y in P_2' are disjoint, say $R(x, P_2') < L(y, P_2')$, whereas in P_2 they intersect. It implies that $R(x, P_2) = L(y, P_2)$ and, by Lemma 8.1.5, H_2 in not quasi-minimal, which is a contradiction. □

Given only the arbitrary graph G and a quasi-minimal interval completion H_2, we know by Theorem 8.1.8 that H_2 is defined by a folding of H_0, a minimal interval completion of G. In general is difficult to find directly the graph H_0. Instead, we can analyze a sequence of interval completions that are between H_0 and H_2, where passing from one step to another needs a folding of a much more constrained nature than the general one – that we call a *reduced folding*.

Definition 8.1.9. *Let $(H_0, \mathcal{Q}_0)$ be a folding. A clique $K \in \mathcal{Q}$ is called a* pivot *in $(H_0, \mathcal{Q}_0)$ if there is a clique-path P_0 of H_0 where both cliques just next to K (one to the left, the other to the right) in P_0 are on the same side of K in $\mathcal{Q}_0$.*

Definition 8.1.10. *A folding $(H, \mathcal{Q})$ is said to be* reduced *if every pivot contains a simplicial vertex of H.*

Given a quasi-minimal interval completion H_2 of H_0, there always is an interval graph H_1 between them (possibly equal H_0), such that H_2 comes from a reduced folding of H_1.

Theorem 8.1.11. *Let H_2 be a quasi-minimal interval completion of H_0 defined by a folding $(H_0, \mathcal{Q}_0)$. Then there is an interval graph H_1 such that $H_0 \subseteq H_1 \subset H_2$,*
and $H_2 = \texttt{FillFolding}(H_1, \mathcal{Q}_1)$ for some reduced folding $(H_1, \mathcal{Q}_1)$ of H_1.

Proof. Recall that H_2 is a quasi-minimal interval completion of H_0 defined by a folding $(H_0, \mathcal{Q}_0)$. Let H_1 be an interval graph such that $H_0 \subseteq H_1 \subset H_2$ and H_1 is inclusion-maximal for this property. Observe that H_2 is a quasi-minimal interval completion of H_1, otherwise H_2 would not be a quasi-minimal interval completion of H_0. By Theorem 8.1.8, there is a folding $(H_1, \mathcal{Q}_1)$ of H_1 such that $H_2 = \texttt{FillFolding}(H_1, \mathcal{Q}_1)$. We claim that $(H_1, \mathcal{Q}_1)$ is reduced.

By contradiction, suppose there is a pivot K of $(H_1, \mathcal{Q}_1)$ containing no simplicial vertex of H_1. Let P_1 be a clique-path of H_1 such that the two cliques next to K in P_1 are on the same side of K in $\mathcal{Q}_1$. Let K' (resp. K'') denote the neighbor of K in P_1 which is closest (resp. furthest) to it in $\mathcal{Q}_1$. Choose $u \in K \setminus K'$ and $v \in K' \setminus K$. The vertices u and v are not adjacent in H_1. Since u is not simplicial in H_1 we must have $u \in K''$. Consequently uv is an edge of H_2: indeed K' is between K and K'' in $\mathcal{Q}_1$, $u \in K \cap K''$, thus in the graph $H_2 = \texttt{FillFolding}(H_1, \mathcal{Q}_1)$, the set $K' \cup \{u\}$ induces a clique. Denote by H the graph obtained from H_1 by adding the edge uv. Add on the clique path P_1, a bag between K and K' containing the vertex subset $K \cap K' \cup \{u, v\}$; we obtain a clique-path of H. In particular H is an interval graph. If $H = H_2$ then H_2 is not a quasi-minimal completion of H_1 – a contradiction. It remains that H is strictly contained in H_2, but also $H_1 \subset H$, contradicting our choice of H_1. □

An important property related to reduced foldings is that each pivot is a core clique.

Lemma 8.1.12. *Let $H_2 = \texttt{FillFolding}(H_1, \mathcal{Q}_1)$ for a reduced folding $(H_1, \mathcal{Q}_1)$. Then every pivot of $(H_1, \mathcal{Q}_1)$ contains a simplicial vertex in H_2, thus the pivot is contained in exactly one maximal clique of H_2.*

Proof. Let x be a simplicial vertex of H_1 contained in the pivot K, so K is the unique maximal clique of H_1 containing x. In the path decomposition of H_2 produced by the algorithm `FillFolding` on $(H_1, \mathcal{Q}_1)$, the unique bag containing x is the bag corresponding to K. Therefore K is contained in a unique maximal clique of H_2. □

8.2 Unfolding

Let H_2 be a quasi-minimal interval completion with a clique-path P_2, obtained by the Algorithm `FillFolding` on $(H_0, \mathcal{Q})$. For the analysis presented in this section, we need to fix a clique-path P_0 of H_0. The reason for this is that with the general definition of a pivot, a pivot K may have different neighbors in distinct clique-paths of H_0. So it may happen that the neighbors of K in P_0 appear at the same side of K in the permutation $\mathcal{Q}$, whereas the neighbors of K in P_0', another clique-path of H_0, appear at different sides of K in $\mathcal{Q}$. For the ease of argument, from now on we should think of a folding as a triple.

Definition 8.2.1. *Let H be an interval graph, P be a clique-path of H and $\mathcal{Q}$ be a permutation of its maximal cliques. The triple $(H, \mathcal{Q}, P)$ is a* folding.

The definition of a pivot becomes more constrained.

Definition 8.2.2. *Let $(H_0, \mathcal{Q}, P_0)$ be a folding. A clique $K \in \mathcal{Q}$ is called a* pivot *in $(H_0, \mathcal{Q}, P_0)$ if both cliques just next to K in P_0 are on the same side of K in $\mathcal{Q}_0$.*

Definition 8.2.3. *Let H_0 be an interval graph with a clique-path P_0. Let $(H_0, \mathcal{Q}, P_0)$ be a reduced folding. If $\mathcal{Q}$ contains just one pivot then it is called a* 1-folding. *If $\mathcal{Q}$ contains exactly 2 pivots, none of which is at an end of $\mathcal{Q}$, then it is called a* 2-folding.

We show in this section that if the quasi-minimal completion H_2 of G is not minimal, there is an interval graph H_1 containing G and strictly contained in H_2 such that H_2 is obtained by a reduced 1-folding or a reduced 2-folding of H_1. In the next section we give a polynomial algorithm constructing H_1.

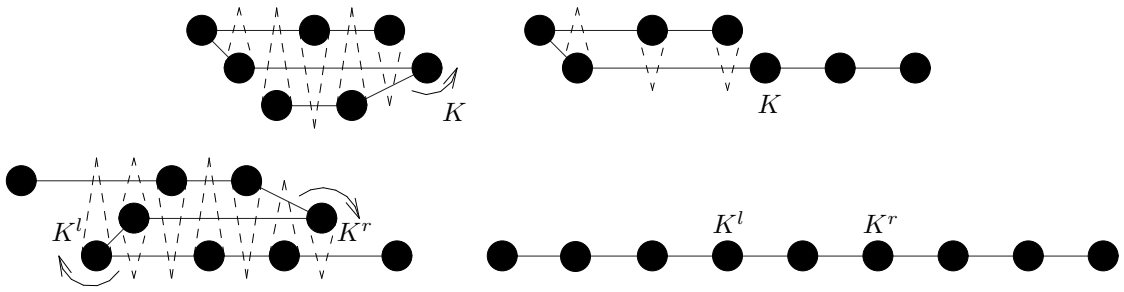


Figure 8.1: Unfolding.

Let us sketch the main idea before getting into the technical proofs. The graph H_2 is equal to $\texttt{FillFolding}(H_0, \mathcal{Q}, P_0)$ for some smaller interval completion H_0 of G and some folding of H_0. Informally, we will slightly *unfold* $(H_0, \mathcal{Q}, P_0)$:

Definition 8.2.4. *A folding $(H_0, \mathcal{Q}', P_0)$ is an* unfolding *of $(H_0, \mathcal{Q}, P_0)$ if the set of pivots of $(H_0, \mathcal{Q}', P_0)$ is strictly contained into the set of pivots of $(H_0, \mathcal{Q}, P_0)$ and, moreover, the graph $\texttt{FillFolding}(H_O, \mathcal{Q}')$ is a (not necessarily strict) subgraph of $\texttt{FillFolding}(H_0, \mathcal{Q})$.*

We will construct an unfolding $(H_0, \mathcal{Q}', P_0)$, having one or two pivots less than $(H_0, \mathcal{Q}, P_0)$. Then H_2 is obtained by a 1 or 2-folding of $H_1 = \texttt{FillFolding}(H_0, \mathcal{Q}')$.

In Figure 8.1 we give examples of unfoldings. Each circle represents a maximal clique in the input interval graph H_0, and each line provides the information that the two maximal cliques are consecutive in the clique path P_0 of H_0. The foldings are defined by the order of the maximal cliques from left to right. Arrows indicate in which direction a sub path is rotated around the pivot when we open a fold. The upper part demonstrates how to unfold when one of the extremal maximal cliques is a pivot, like K in this case. The lower part shows how to unfold when K^r is the rightmost pivot in $\mathcal{Q}$ and K^l is the leftmost pivot in $\mathcal{Q}$ that appears after K^r in P_0.

The following theorem states that it is always possible to unfold a quasi-minimal interval completion.

Theorem 8.2.5. *Let H_2 be a quasi-minimal, but not minimal interval completion of an arbitrary graph G. Then there exits a reduced folding $(H_1, \mathcal{Q}_1, P_1)$, with one (1-folding) or two (2-folding) pivots, with $E(G) \subseteq E(H_1) \subset E(H_2)$ and such that $H_2 = \texttt{FillFolding}(H_1, \mathcal{Q}_1)$.*

Proof. We know from Theorem 8.1.11 that there exists a reduced folding $(H_0, \mathcal{Q}, P_0)$ satisfying $H_2 = \texttt{FillFolding}(H_0, \mathcal{Q})$ and $E(G) \subseteq E(H_0) \subset E(H_2)$. We may assume that for any unfolding $(H_0, \mathcal{Q}', P_0)$, the graph $H_2' = \texttt{FillFolding}(H_0, \mathcal{Q}')$ is strictly contained in H_2.

We distinguish two different cases, as depicted in Figure 8.1. The maximal cliques of H_0 are numbered according to their order in the path P_0. The permutation $\mathcal{Q}$ corresponds to the ordering of the maximal cliques from left to right. The permutation $\mathcal{Q}$ is transformed in the permutation $\mathcal{Q}'$, as depicted below.

First case: a maximal clique at one end of $\mathcal{Q}$ is a pivot. Choose this as the left end and call the maximal clique K. (see the upper part of Figure 8.1.) Clearly, K is not one of the end maximal cliques in P_0. Let P^l be the subpath of P_0, from the left end to the clique K (included) and P^r be the subpath from K (included) to the right end of P_0. Let $\mathcal{Q}^l$ be the order defined by $\mathcal{Q}$ on

the maximal cliques of P^l, and let $\mathcal{Q}^r$ be the order on the maximal cliques of P^r. Glue the reverse order of $\mathcal{Q}^l$ with $\mathcal{Q}^r$ in K (i.e, concatenate the reverse of $\mathcal{Q}^l$ with $\mathcal{Q}^r$ and merge the two consecutive copies of K) to obtain the new permutation $\mathcal{Q}'$ and the folding $(H_0, \mathcal{Q}', P_0)$.

Let us now argue that $(H_0, \mathcal{Q}', P_0)$ is an unfolding of $(H_0, \mathcal{Q}, P_0)$. The pivot K is not a pivot of the new folding, and clearly every pivot of $(H_0, \mathcal{Q}', P_0)$ is a pivot of $(H_0, \mathcal{Q}, P_0)$. We show that $H_1 = \texttt{FillFolding}(H_0, \mathcal{Q}')$ is a strict subgraph of H_2. Let V^l and V^r be the union of maximal cliques in $\mathcal{Q}^l$ and $\mathcal{Q}^r$, let $H^l = \texttt{FillFolding}(H[V^l], \mathcal{Q}^l)$ and $H^r = \texttt{FillFolding}(H[V^r], \mathcal{Q}^r)$. Clearly, H^l and H^r are subgraphs of H_2. The maximal clique K of H_0 separates, in the graph H_0, the vertex subsets V^l and V^r. Hence it also separates V^l and V^r in H_1; in particular K is a maximal clique of H_1. Each edge of H_1 is an edge of H^l or of H^r, thus the graph H_1 is a subgraph of H_2. By our assumption, the graph H_1, obtained by an unfolding of $(H_0, \mathcal{Q}, P_0)$, is a strict subgraph of H_2.

It remains to prove that there exists a reduced 1-folding $(H_1, \mathcal{Q}_1, P_1)$ of H_1 such that $H_2 = \texttt{FillFolding}(H_1, \mathcal{Q}_1)$. Consider the path decomposition PD_1 of H_1 produced by the algorithm `FillFolding` on $(H_0, \mathcal{Q}')$. The bags of PD_1 are in one-to-one correspondence with the maximal cliques of H_0, as they appear in the permutation $\mathcal{Q}'$. In particular, the original permutation $\mathcal{Q}$ induces a permutation $\mathcal{QB}_1$ on the bags of PD_1. Let Q_1 be obtained from $\mathcal{QB}_1$ as the sublist of bags corresponding to maximal cliques of H_1. The clique-path P_1 is constructed by removing the bags of PD_1 which are not maximal cliques of H_1. By construction, $(H_1, \mathcal{Q}_1, P_1)$ is a folding of H_1 having K as unique fold, in particular it is a reduced folding. We show that $H_2 = \texttt{FillFolding}(H_1, \mathcal{Q}_1)$. Suppose that we apply the algorithm `FillFolding` on $(H_1, \mathcal{QB}_1)$ (although $\mathcal{QB}_1$ is not a permutation of the set of maximal cliques of H_1). We obtain the same graph as $\texttt{FillFolding}(H_0, \mathcal{Q}_0)$. Indeed, assign to each vertex u the interval $[s_{\mathcal{QB}_1}(u), t_{\mathcal{QB}_1}(u)]$, where $s_{\mathcal{QB}_1}(u)$ (resp. $t_{\mathcal{QB}_1}(u)$) is the minimum (resp. maximum) index of a bag of $\mathcal{QB}_1$ containing u. This interval is the same as $[s_{\mathcal{Q}}(u), t_{\mathcal{Q}}(u)]$, obtained by the same construction applied to the permutation $\mathcal{Q}$. Now, by our construction of $\mathcal{Q}_1$, it can be easily checked that the graph $\texttt{FillFolding}(H_1, \mathcal{Q}_1)$ is the same as $\texttt{FillFolding}(H_1, \mathcal{QB}_1)$. Hence $H_2 = \texttt{FillFolding}(H_1, \mathcal{Q}_1)$.

Second case: neither of the maximal cliques at the ends of $\mathcal{Q}$ are pivots. Let K^l be the leftmost pivot, according to the ordering $\mathcal{Q}$ (see the lower part of Figure 8.1). Since no pivot of $\mathcal{Q}$ is the left or right most maximal clique, then there exists exactly one end of P_0 that contains maximal cliques to the right of K^l in $\mathcal{Q}$, let this be the right end of P_0. Let K^r be the rightmost pivot in $\mathcal{Q}$, such that K^l is to the right of K^r in P_0. Let P^l be the subpath of P_0, from the left end to the clique K^r included (we remind that K^r appears left to K^l in P_0). Similarly P^r is the subpath of P_0 from K_l (included) to the right end, and P^c is the subpath of P_0 starting in K^r and ending in K^l. Let $\mathcal{Q}^l$ (resp. $\mathcal{Q}^r$, $\mathcal{Q}^c$) be the order defined by $\mathcal{Q}$ on the maximal cliques of P^l (resp. P^r, P^c). Glue the reverse order of $\mathcal{Q}^r$ with $\mathcal{Q}^c$ in K_l and glue this with the reverse order of $\mathcal{Q}^l$ in K^r, and obtain the new order $\mathcal{Q}''$ and folding $(H_0, \mathcal{Q}'', P_0)$. Ir remains to show that the unfolding $(H_0, \mathcal{Q}'', P_0)$ satisfies the required properties.

Like in the first case, $(H_0, \mathcal{Q}'', P_0)$ is an unfolding of $(H_0, \mathcal{Q}, P_0)$. Each pivot of $(H_0, \mathcal{Q}'', P_0)$, except for K^l and K^r, is a pivot in $(H_0, \mathcal{Q}, P_0)$. Indeed such a pivot is contained in $\mathcal{Q}^l$, $\mathcal{Q}^c$, or $\mathcal{Q}^r$ and the maximal cliques in $\mathcal{Q}^l$, $\mathcal{Q}^c$ and $\mathcal{Q}^r$ induce connected sub paths of P_0. Let V^l, V^c, V^r be the union of maximal cliques in $\mathcal{Q}^l$, $\mathcal{Q}^c$, and $\mathcal{Q}^r$, and let H^l, H^c and H^r be the graphs $\texttt{FillFolding}(H[V^l], \mathcal{Q}^l)$, $\texttt{FillFolding}(H[V^c], \mathcal{Q}^c)$ and $\texttt{FillFolding}(H[V^r], \mathcal{Q}^r)$ respectively. $H_1 = \texttt{FillFolding}(H_0, \mathcal{Q}'')$ is a subgraph of H_2, since every edge and vertex of H_1 is contained in either H^l, H^c, or H^r, and $H^l = H_2[V^l], H^c = H_2[V^c]$, and $H^r = H_2[V^r]$. We have assumed that for any unfolding of

$(H_0, \mathcal{Q}, P_0)$, the filled graph is a strict subgraph of H_2. Consequently H_1 is a strict subgraph of H_2.

We prove that there exists a reduced 2-folding $(H_1, \mathcal{Q}_1, P_1)$ defining the graph H_2 (i.e., $H_2 = \texttt{FillFolding}(H_1, \mathcal{Q}_1)$). The construction is very similar to the first case. Let PD_1 be the path decomposition of H_1 produced by the algorithm `FillFolding` on $(H_0, \mathcal{Q}'')$. Its bags are in one-to-one correspondence with the maximal cliques of H_0, as they appear in the permutation $\mathcal{Q}''$. Let $\mathcal{QB}_1$ be the permutation of these bags corresponding to the original permutation $\mathcal{Q}$, and $\mathcal{Q}_1$ be the sublist of $\mathcal{QB}_1$ formed by the bags corresponding to maximal cliques of H_1. The clique-path P_1 is obtained from PD_1 by removing the bags which are not maximal cliques of H_1. The folding $(H_1, \mathcal{Q}_1, P_1)$ has only two folds, namely K^l and K^r. As in the first case, one can check that $H_2 = \texttt{FillFolding}(H_1, \mathcal{QB}_1)$, and eventually $\texttt{FillFolding}(H_1, \mathcal{Q}_1)$ yields the same graph as $\texttt{FillFolding}(H_1, \mathcal{QB}_1)$. Consequently $(H_1, \mathcal{Q}_1, P_1)$ is a reduced 2-folding of H_1 such that $H_2 = \texttt{FillFolding}(H_1, \mathcal{Q}_1)$.

□

The first observation gives some information on the structure of 1-folding that will be used in the unfolding algorithm.

Observation 8.2.6. *Let $(H_1, \mathcal{Q}_1, P_1)$ be a reduced 1-folding and let $H_2 = \texttt{FillFolding}(H_1, \mathcal{Q}_1)$. Then its pivot is a maximal clique in H_2. Moreover, there is a clique path of H_2 such that this pivot corresponds to a leaf.*

Proof. Since $(H_1, \mathcal{Q}_1, P_1)$ is a 1-folding, there is only one pivot K in $\mathcal{Q}_1$ and clearly K is at one end of $\mathcal{Q}_0$. By Algorithm `FillFolding`, no vertex is added to K and K is at the end of the path decomposition P_2 produced by the algorithm. Because the folding is reduced, there is a simplicial vertex x of H_1 such that the only bag of P_1 (and therefore of P_2) containing x is K. Consequently K is a maximal clique of H_2, corresponding to a leaf of P_2. □

And the other observation covers the case of 2-folding. Like the previous one, it says that the clique path of H_2 obtained by folding keeps much of the structure of the original clique path of H_1.

Observation 8.2.7. *Let $(H_1, \mathcal{Q}, P_1)$ be a 2-folding that defines $H_2 = \texttt{FillFolding}(H_1, \mathcal{Q})$, and let P_2 be the clique path of H_2 obtained by the folding algorithm. Let K^r (K^l) be the leftmost (rightmost) pivot according to the ordering P_1 (hence K^l appears before K^r in $\mathcal{Q}$). Let P^l be the subpath of P_1 formed by the cliques appearing strictly before K^l in $\mathcal{Q}$ and P^r be the subpath of P_1 formed by the cliques appearing strictly after K^r in $\mathcal{Q}$. Then P^l and P^r are end paths of both P_1 and P_2.*

Proof. It follows directly from Algorithm `FillFolding` applied to $(H_1, \mathcal{Q})$ that P^l and P^r are the end clique paths of P_2 as well. □

The next results state that, if H_2 is a quasi-minimal interval completion of G and comes from a 1-folding or 2-folding of some $H_1 \subset H$, there is an edge uv in $E(H_2) \setminus E(H_1)$ with special properties. In the next section, we shall ensure that the unfolding algorithm removes this edge.

Theorem 8.2.8. *Let $H_1 = (V, E_1)$, $H_2 = (V, E_2)$ be interval graphs and let $(H_1, \mathcal{Q}, P_1)$ be a 1 or 2-folding such that $H_2 = \texttt{FillFolding}(H_1, \mathcal{Q})$. If H_2 is a quasi-minimal but not minimal interval completion of H_1, then there is a fill edge uv, such that one of the pivots K is a u,v-separator in H_1. Moreover, in the clique path P_1, the vertices u and v appear on different sides of K.*

Proof. By Observation 8.2.7, if P_2 is the path decomposition of $H_2 = \texttt{FillFolding}(H_1, \mathcal{Q})$ obtained by the folding algorithm on a 2-folding $(H_1, \mathcal{Q}, P_1)$, we can write

$$P_1 = P^l - -P_1^l - -K^r - -P_1^c - -K^l - -P_1^r - -P^r \tag{8.1}$$

$$P_2 = P^l - -\Omega^l - -P_2^c - -\Omega^r - -P^r \tag{8.2}$$

Notice that a 1-folding can be treated as a 2-folding with one side of P_1, say $P_1^r - -P^r$, empty. In such a situation, K^l is not a pivot according to the previous definition, but if we extend it and define K^l to be a pivot, then the following argument holds for the case of 1-folding as well.

Suppose, on the contrary, that no pivot is a x, y-separator in H_1 for any fill edge xy. Let us first construct a clique path P_2' of H_2 based on P_1. We will use it later to prove that H_2 is not quasi-minimal.

Let P_2 be the clique path obtained by Algorithm `FillFolding` on $(H_1, \mathcal{Q})$ (see Figure 7.1), so P_1 and P_2 are described by Equations 8.1, 8.2. Let $PL = P^l - -P_1^l - -K^r$, $PC = K^r - -P_1^c - -K^l$ and $PR = K^l - -P_1^r - -P^r$. Let xy be any fill edge with $x \in \mathrm{V}(PL) \setminus K^r$, then $y \in K^r$, since K^r is not a x, y-separator. So $N_{H_2}(\mathrm{V}(PL) \setminus K^r) \subseteq \mathrm{V}(PL)$. By Algorithm `FillFolding`, the bags of PL that have y added in the folding $(H_1, \mathcal{Q})$ form an interval. We add y to the bags of this interval, for every such fill edge xy. In this way we simulate the folding $(H_1, \mathcal{Q})$ on $H_1[\mathrm{V}(PL)]$ and obtain a clique path PL' of $H_2[\mathrm{V}(PL)]$. In an analogous way, we simulate the folding on PR and PC to obtain PR' and PC', clique paths of $H_2[\mathrm{V}(PR)]$ and $H_2[\mathrm{V}(PC)]$, respectively. Altogether, by gluing the reverse of PL' with PC' and the reverse of PR' on K^l and K^r, (and possibly removing redundant bags) we obtain a clique path P_2' of H_2. Indeed, $N_{H_2}(\mathrm{V}(PL) \setminus K^r) \subseteq \mathrm{V}(PL)$, $N_{H_2}(\mathrm{V}(PC) \setminus (K^r \cup K^l)) \subseteq \mathrm{V}(PC)$ and $N_{H_2}(\mathrm{V}(PR) \setminus K^l) \subseteq \mathrm{V}(PR)$, so every edge of H_2 is contained in some bag of P_2'. Moreover, it is easy to verify that, for every vertex $x \in V$, the subset of bags containing x induces a subpath of P_2'.

Suppose $H_2[\mathrm{V}(PL')]$ contains a fill edge $x'y'$ with $y' \in K^r$. Let Ω be the maximal clique containing y' leftmost in P_2'. There is a vertex x'' which does not appear in a bag right of Ω, since Ω is a maximal clique. Moreover, $x''y' \notin E(H_1)$, since $x''y' \in E(H_1)$ implies $x'y' \in E(H_1)$. Notice that $\Omega \neq K^r$, since in $H_2[\mathrm{V}(PL')]$ K^r is a maximal clique in which does not contain any fill edges. Indeed, by definition of reduced folding, K^r contains a vertex simplicial in H_2. Since Ω is the leftmost bag containing y', there is a clique $\Omega_{x'y'}$ containing both x' and y' between Ω and K^r in P_2'. By construction of P_2', if $x''y' \in E(H_1)$ then x' and y' appear together in a corresponding bag $K_{x'y'}$ of P_1 as well. So $x''y'$ is a fill edge. Let P_2'' be the clique path obtained from P_2' by removing Ω and putting $(\Omega \setminus \{y'\}) - -(\Omega \setminus \{x''\})$ instead. Clearly, it is a clique path of $H_2 - x''y'$, hence H_2 is not a quasi-minimal interval completion of H_1 - a contradiction. So $H_2[\mathrm{V}(PL')]$ does not contain any fill edges. An analogous argument shows that there is no fill edge in $H_2[\mathrm{V}(PR')]$.

Finally, suppose only $H_2[\mathrm{V}(PC')]$ contains a fill edge $x'y'$. So one of the vertices, say y' is contained in one of the pivots, say K^r, in H_1. Notice that y' is not contained in K^l, otherwise, by properties of clique path P_1, $x'y'$ would be an edge already in H_1. So, again, we pick Ω to be the rightmost in PC' maximal clique containing y' and a vertex x'' which does not appear in a bag left of Ω. Notice that $N_{H_2}(y') \cap \mathrm{V}(PR') = \emptyset$, since y' is not contained in K^l and there are no fill edges in $H_2[\mathrm{V}(PR')]$. Let P_2'' be the clique path obtained from P_2' by removing Ω and putting $(\Omega \setminus \{x''\}) - -(\Omega \setminus \{y'\})$ instead. Clearly, it is a clique path of $H_2 - x'y''$, which contradicts the quasi-minimality of H_2. A contradiction. □

8.3 Extracting minimal interval completions: the algorithm

Let H_2 be a quasi-minimal interval completion of G and let H_0 be a minimal interval completion of G contained in H_2. Theorem 8.1.8 shows that there exists a one folding $(H_0, \mathcal{Q}_0, P_0)$ that defines H_2, and by Theorem 8.2.5 there exists a reduced folding $(H_1, \mathcal{Q}_1, P_1)$ with one or two pivots that defines H_2. By Lemma 8.1.12 each pivot of $(H_1, \mathcal{Q}_1, P_1)$ is contained in exactly one maximal clique of H_2.

Let us now assume that $(H_1, \mathcal{Q}_1, P_1)$ is such that any unfolding defines a graph with fewer edges than H_2. We will focus of finding an unfolding such that some fill edge uv is removed. The edge uv is chosen such that one of the pivots of $(H_1, \mathcal{Q}_1, P_1)$ is a u, v-separator in H_1 (see Theorem 8.2.8).

We will consider the cases of 1-folding and 2-folding separately. Let us first discuss the 1-folding case. Remember from Observation 8.2.6 that a maximal clique K of H_2 is a pivot in P_1 if $(H_1, \mathcal{Q}_1, P_1)$ is an 1-folding defining H_2.

Algorithm `OneUnfolding`
Input: A graph $G = (V, E)$, and an interval completion H_2 of G
Output: An interval completion H_1' of G such that
$E(H_1') \subset E(H_2)$ if H_2 is defined by a 1-folding of some H_1
$H_1' = H_2$ if no H_1 exists.

for each pair (Ω, u) where $\Omega \in \mathcal{K}(H_2)$ and $u \in V \setminus \Omega$
  Let C_u be connected comp of $G[V \setminus \Omega]$ containing u
  $H_1' = (V, E(H_2[N_G[C_u]]) \cup E(H_2[V \setminus C_u]))$
  if H_1' is an interval graph and $E(H_1') \subset E(H_2)$ then
    return H_1'
return H_2

Figure 8.2: Opening one pivot.

Lemma 8.3.1. *Let $G = (V, E)$ be an arbitrary graph, and let H_1 and H_2 be two interval completions of G, such that $E(H_1) \subset E(H_2)$, H_2 is a quasi-minimal interval completion of H_1, and $(H_1, \mathcal{Q}_1, P_1)$ is a 1-folding that defines H_2. Then $H_1' =$ `OneUnfolding`(G, H_2) is an interval completion of G satisfying $E(H_1') \subset E(H_2)$.*

Proof. Algorithm `OneUnfolding` always outputs an interval graph H_1' such that $E(G) \subseteq E(H_1') \subseteq E(H_2)$. We have to prove that, under the conditions of the lemma, the graph H_1' is a strict subgraph of H_2.

From Observation 8.2.6 it follows that the pivot in $(H_1, \mathcal{Q}_1, P_1)$ is a maximal clique in H_2; let this maximal clique be Ω. Moreover, Ω is at one end, say the left one, of a clique path P_2 of H_2. Let uv be one of the edges in $E(H_2) \setminus E(H_1)$, such that Ω is a u, v-separator in H_1 (see Theorem 8.2.8). In particular, both $u, v \notin \Omega$.

Like in Algorithm `OneUnfolding`, let C_u be the connected component of $G[V \setminus \Omega]$ containing the vertex u and consider the graph $H_1' = (V, E(H_2[N_G[C_u]]) \cup E(H_2[V \setminus C_u]))$. We have to argue that $E(H_1') \subset E(H_2)$ and that H_1' is an interval graph. First $E(H_1') \subseteq E(H_2)$ since only edges in H_2 are used to create H_1'. Since Ω is a u, v-separator in H_1, we have $v \notin N_G[C_u]$. Clearly $u \notin V \setminus C_u$, so uv is not an edge of H_1'.

Let us construct a path decomposition of H_1' such that each bag induces a clique. This shows that H_1' it is indeed an interval graph. Take P_2 and glue it in Ω with P_2', an reversed copy of P_2. Remove vertices from bags in order to have only the vertices of $V \setminus C_u$ in the P_2 part and the vertices of $N_G[C_u]$ in the P_2' part. Let P_1' denote the result. Clearly, it is a path decomposition of H_1'. By removing redundant bags we obtain a clique-path of H_1'. □

Algorithm `TwoUnfolding`
Input: A graph $G = (V, E)$, and an interval completion H_2 of G
Output: An interval completion H_1' of G such that
$E(H_1') \subset E(H_2)$ if H_2 is defined by a 2-folding of some $H_1 \subset H_2$
$H_1' = H_2$ if no H_1 exists.

for each tuple $(\Omega^l, \Omega^r, S^l, S^r, C^l, C^r, u, v)$
  # Ω^l, Ω^r are maximal cliques of H_2
  # S^l, S^r are minimal separators of H_2, contained in Ω^l and resp. Ω^r.
  # C^l (C^r) is a component of $H_2 - S^l$ (resp. $H_2 - S^r$)
  # u, v are vertices, $u \notin \Omega^r$
  construct W^l using Equation 8.4
  construct W^r using Equation 8.5
  $H_1' = (V, E(H_2[N_G[W^l] \cup S^l]) \cup E(H_2 - (W^l \cup W^r)) \cup E(H_2[N_G[W^r] \cup S^r]))$
  if H_1' is an interval graph and $E(H_1') \subset E(H_2)$ then
    return H_1'
return H_2

Figure 8.3: Opening two pivots.

A sketch of the algorithm is given in Figure 8.3, and its correctness is stated below.

Lemma 8.3.2. *Let $G = (V, E)$ be an arbitrary graph, and let H_1 and H_2 be two interval completions of G, such that $E(H_1) \subset E(H_2)$, H_2 is a quasi-minimal interval completion of H_1, and $(H_1, \mathcal{Q}_1, P_1)$ is a 2-folding that defines H_2. Then $H_1' =$ `TwoUnfolding`(G, H_2) is an interval completion of G satisfying $E(H_1') \subset E(H_2)$.*

The proof of this lemma is quite technical. Let us postpone it to a separate section. Now we can state the main result of this chapter.

Lemmas 8.3.1 and 8.3.2 imply the main result of this paper. Algorithm `ExtractMinimalIntervalCompletion` is given in Figure 8.4.

Theorem 8.3.3. *There exists a polynomial time algorithm that, given an arbitrary graph G and an interval completion H_2 of G, computes a minimal interval completion H_1 of G, such that $E(H_1) \subseteq E(H_2)$.*

Proof. We show that Algorithm `ExtractMinimalIntervalCompletion` of Figure 8.4 computes the required completion. At each step of our algorithm the graph H_1 is an interval graph, by construction of Algorithms `OneUnfolding` and `TwoUnfolding`. Also $E(G) \subseteq E(H_1) \subseteq E(H_2)$. It remains to show that, at the end of our algorithm, H_1 is a minimal interval completion of G. Otherwise,

Algorithm `ExtractMinimalIntervalCompletion`
Input: A graph $G = (V, E)$, and an interval completion H_2 of G.
Output: A minimal interval completion H_1 of G, with $E(H_1) \subseteq E(H_2)$.

$H_1 = H_2$
$H_0 = G$
while $(H_0 \neq H_1)$
    $H_0 = H_1$
    for each edge uv in $E(H_1) \setminus E(G)$
        if $H_1 - uv$ is an interval graph then
            $H_1 = H_1 - uv$
    $H_1 = \texttt{OneUnfolding}(G, H_1)$
    $H_1 = \texttt{TwoUnfolding}(G, H_1)$
return H_1

Figure 8.4: Extracting a minimal interval completion.

H_1 is not quasi-minimal or, by Theorem 8.2.5, there exists a strict subgraph H_1' such that H_1' is an interval completion of G and H_1 corresponds to a reduced 1-folding or 2-folding of H_1'. By Lemmas 8.3.1 and Lemma 8.3.2, Algorithms `OneUnfolding` or `TwoUnfolding` would construct an interval completion of G, strictly included in H_1. This contradicts the fact that we finished the **while** loop.

Since an interval graph has at most n maximal cliques and minimal separators, Algorithms `OneUnfolding` and `TwoUnfolding` are clearly polynomial. At each iteration of the **while** loop, our algorithm removes at least one edge. We conclude that Algorithm `ExtractMinimalIntervalCompletion` terminates in polynomial time. □

Let us point out that, by using as initial completion the complete graph, the algorithm `ExtractingMinimalIntervalCompletion` can obtain any of the minimal interval completions of G.

8.4 Proof of Lemma 8.3.2

Proof. For better reading, we re-discuss this case completely.

Let $H_2 = (V, E_2)$ be an quasi-minimal interval completion of a non-interval graph $G = (V, E)$. Suppose that H_2 is not minimal and choose the graph $H_1 = (V, E_1)$ such that $H_2 = \texttt{FillFolding}(H_1, \mathcal{Q})$, where $(H_1, \mathcal{Q}, P_1)$ is a reduced 2-folding and $E \subset E_1 \subset E_2$. Let P_2 be the clique path of H_2 obtained by the algorithm $\texttt{FillFolding}(H_1, P_1)$.

So by Observation 8.2.7 we can denote these clique paths as:

$$P_1 = P^l - -P_1^l - -K^r - -P_1^c - -K^l - -P_1^r - -P^r$$

$$P_2 = P^l - -\Omega^l - -P_2^c - -\Omega^r - -P^r$$

where K^l, K^r are the pivots and Ω^r, Ω^l are the maximal cliques of H_2 containing them. We have $K^l \subseteq \Omega^l, K^r \subseteq \Omega^r$.

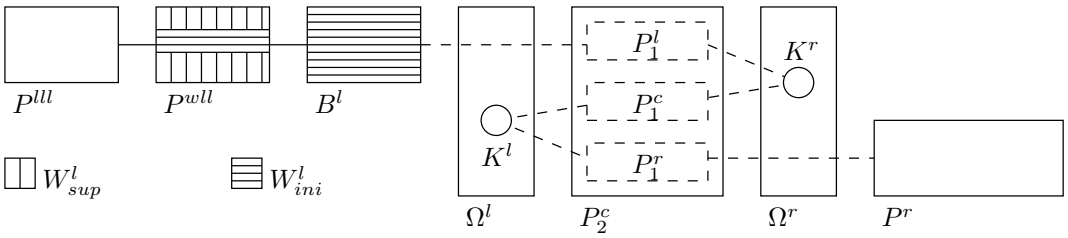


Figure 8.5: 2-folding.

Let S^l (S^r) be the separator between P^l and Ω^l (Ω^r and P^r) in P_2. Notice that P^l, S^l, S^r, P^r appear also in P_1. Let B^l (B^r) be the interval of cliques that corresponds to the block of $H_2 - S^l$ ($H_2 - S^r$) that is contained in P^l (P^r), the closest to Ω^l (Ω^r) in P_2. So we have:

$$P_2 = P^{ll} - -B^l - -\Omega^l - -P_2^c - -\Omega^r - -B^r - -P^{rr} \tag{8.3}$$

Notice that $C^l = \mathrm{V}(B^l) \setminus \Omega^l$ ($C^r = \mathrm{V}(B^r) \setminus \Omega^r$) is a connected component of $H_2 - \Omega^l$ ($H_2 - \Omega^r$). So, given H_2, Ω^l and Ω^l, we can effectively compute the candidates for C^l and C^r. From now on we assume that $\Omega^l, \Omega^r, C^l, C^r, S^l, S^r$ are as described above. We want to find an unfolding of H_2, an interval graph $H_1' = (V, E(H_2[N_G[W^l] \cup S^l]) \cup E(H_2[N_G[W^r] \cup S^r]) \cup E(H_2 - (W^l \cup W^r)))$, with some well chosen W^l, W^r.

Let uv be an edge of $E(H_2) \setminus E(H_1)$. Like in Theorem 8.2.8, u and v are chosen such that one of the pivots of H_1, say K^r, separates u and v in H_1 and also in the clique path P_1. Suppose w.l.o.g. that u is in $\mathrm{V}(P^l - -P_1^l) \setminus K^r$. In particular $u \notin \Omega^r$, and let C_u be the connected component of $G - \Omega^r$ containing the vertex u. Let $\mathcal{C}_v$ be the union of the connected components of $G - \Omega^r$ that contain or see the vertex v (we may have $v \in \Omega^r$, in which case there are several such components). We have proved:

Claim 10. *C_u is contained in $\mathrm{V}(P^l - -P_1^l) \setminus K^r$. $\mathcal{C}_v$ is contained in $\mathrm{V}(P_1^c - -K^l - -P_1^r - -P^r)$. Moreover, u and v do not appear in K^r.*

Definition 8.4.1.

$$W^l_{ini} = \bigcup \{C \mid C \in C(G - \Omega^r), C \cap B^l \neq \emptyset\} \cup C_u$$

$$W^r_{ini} = \bigcup \{C \mid C \in C(G - \Omega^l), C \cap B^r \neq \emptyset\}$$

When unfolding, we want the blocks corresponding to components of $H_2 - S^l$ that are in $P^{ll} - -B^l$ to stay blocks in $H_1' - S^l$. Let us investigate the connected components of $H_2 - S^l$. They induce a partition of $P^{ll} - -B^l$ into blocks of $H_2 - S^l$. Some of these blocks intersect connected components of $G - \Omega^r$ that are contained in W^l_{ini}. For example, all connected components of $G - \Omega^r$ that intersect B^l are also in W^l_{ini}. But there may be a block B^{lm} of $H_2 - S^l$ containing some components of $G - \Omega^r$ that are and some that are not in W^l_{ini}. In this case, B^{lm} is not a block of $H_1' - S^l$, where $H_1' = (V, E(H_2[N_G[W^l_{ini}]]) \cup E(H_2[N_G[W^r_{ini}]]) \cup E(H_2[V \setminus (W^l_{ini} \cup W^r_{ini})]))$. In order to prevent this, we augment W^l_{ini} with W^l_{sup}, and W^l_{ini} with W^l_{sup} as defined below:

Definition 8.4.2.

$$W^l_{sup} = \bigcup \{C \mid C \in C(G - \Omega^r), C \cap \Omega^l = \emptyset, N_{H_2}[C] \cap W^l_{ini} \neq \emptyset, N_{H_2}[C] \cap (W^r_{ini} \cup \mathcal{C}_v) = \emptyset\}$$

$$W^l = W^l_{ini} \cup W^l_{sup} \tag{8.4}$$

$$W^r_{sup} = \bigcup \{C \mid C \in C(G - \Omega^l), C \cap \Omega^r = \emptyset, N_{H_2}[C] \cap W^r_{ini} \neq \emptyset, N_{H_2}[C] \cap W^l = \emptyset\}$$

$$W^r = W^r_{ini} \cup W^r_{sup} \tag{8.5}$$

We are now able to construct the unfolding.

Definition 8.4.3.

$$H'_1 = (V, E(H_2[N_G[W^l] \cup S^l]) \cup E(H_2[N_G[W^r] \cup S^r]) \cup E(H_2 - (W^l \cup W^r))).$$

Clearly, $B^l \subseteq N_G[W^l_{ini}] \cup S^l$, $B^r \subseteq N_G[W^r_{ini}] \cup S^r$. Moreover, $W^l_{ini} \subseteq \mathrm{V}(P^l - -P^l_1) \setminus K^r$ and $W^r_{ini} \subseteq \mathrm{V}(P^r_1 - -P^r) \setminus K^l$, so $W^l_{ini} \cap W^r_{ini} = \emptyset$. Notice that by similar argument $N(W^l_{ini}) \cap W^r_{ini} = \emptyset$ and $N(W^r_{ini}) \cap W^l_{ini} = \emptyset$. Moreover, by construction of $W^l \cap W^r$, this propagates to W^l and W^r:

Claim 11. *$W^l \cap W^r = \emptyset$.*

Let us now put some auxiliary notations.

Definition 8.4.4. *Let $W^c = (\Omega^l \cup \mathrm{V}(P^c_2) \cup \Omega^r) \setminus (W^l \cup W^r)$. Let $W^{ll} = \mathrm{V}(P^{ll})$ and $W^{rr} = \mathrm{V}(P^{rr})$.*

Let us construct a path decomposition of the graph H'_1 in which all bags are cliques, let us call such a decomposition a *good path decomposition*, by first constructing good path decompositions of $H_2[N_G[W^{ll} \cup W^l]]$, $H_2[N_G[W^c]]$, $H_2[N[W^r \cup W^{rr}]]$, and then gluing them together.

Let us further refine the description of P^{ll} and denote by P^{lll} the subsequence of P^{ll} induced by blocks of $H_2 \setminus S^l$ that have empty intersection with W^l, an by P^{wll} the subsequence induced by blocks of $H_2 \setminus S^l$ contained in $N_G[W^l] \cup S^l$. The cliques contained in $N_G[W^l] \cup S^l$ are consecutive indeed. Take a clique Ω_2, $\Omega_2 \cap W^l_{ini} = \emptyset$, with another clique Ω_1, $\Omega_1 \cap W^l_{ini} \neq \emptyset$, to the left of it in P_2. Take $x \in \Omega_1 \cap W^l_{ini}$ and C_x, with $C_x \in C(G - \Omega^r), C_x \cap C^l \neq \emptyset, x \in C_x$. Since Ω_2 separates Ω_1 from B^l, it intersects C_x. Therefore $\Omega_2 \cap W^l_{ini} \neq \emptyset$ and $\Omega_2 \subseteq N_G[W^l] \cup S^l$.

Claim 12. *$P^{wl} = P^{lll} - -P^{wll} - -B^l - -P^{lc}_2$, where P^{lc}_2 comes from P^c_2 by removing from every bag the vertices not in $N_G[W^l]$, is a good path decomposition of $H_2[N_G[W^{ll} \cup W^l]]$.*

It follows the construction that $\mathrm{V}(P^{wll} - -B^l - -P^{lc}_2)$ is contained in $N_G[W^l] \cup S^l$. Moreover, $\mathrm{V}(P^{lll})$ is contained in $V \setminus (W^l \cup W^r)$. Indeed, by construction, there is: $\mathrm{V}(P^{lll}) \cap W^l = \emptyset$, $\mathrm{V}(P^{lll}) \cap W^r_{ini} = \emptyset$, $W^r_{sup} \subseteq \mathrm{V}(P^c_2 - -\Omega^r - -P^r) \setminus \Omega^r$, and $\mathrm{V}(P^{lll}) \cap S^l \subseteq \Omega^r$. So $\mathrm{V}(P^{lll}) \cap (W^l \cup W^r) = \emptyset$ and the corresponding cliques of H_2 remain cliques of H'_1. By construction, $N_G[W^{ll} \cup W^l] = \mathrm{V}(P^{wl})$. Moreover, for every $x \in \mathrm{V}(P^{wl})$, the cliques containing it induce an interval in P^{wl}, since they did in $P^{ll} - -B^l - -P^c_2$. In a similar manner, $P^{wr} = P^{rc}_2 - -B^r - -P^{wrr} - -P^{rrr}$ is a good path decomposition of $H_2[N_G[W^r \cup W^{rr}]]$.

Claim 13. *$P^{wr} = P^{rc}_2 - -B^r - -P^{wrr} - -P^{rrr}$, where P^{rc}_2 comes from P^c_2 by removing from every bag the vertices not in $N_G[W^r]$, is a good path decomposition of $H_2[N_G[W^{rr} \cup W^r]]$.*

Finally, let P^{cc}_2, Ω^{cl}, Ω^{cr} come from P^c_2, Ω^l, Ω^r by removing the vertices in $W^l \cup W^r$. Then, by reasoning similar to that above, we have:

Claim 14. *$\Omega^{cl} - -P^{cc}_2 - -\Omega^{cr}$ is a good path decomposition $H_2[W^c]$.*

Finally, we can glue these three good path decompositions together to obtain:

Claim 15. *H'_1 is an interval graph with a good path decomposition*

$$P'_1 = \overleftarrow{P^{wr}} - -\Omega^{cl} - -P_2^{cc} - -\Omega^{cr} - -\overleftarrow{P^{wl}},$$

where the arrow indicates that the corresponding path has been reversed.

By construction and discussion above, the vertices of W^r appear only in bags of P^{wr} and the vertices of W^l appear only in bags of P^{wl}. Moreover $N_G(W^l) \cap W^r = \emptyset$ and $N_G(W^r) \cap W^l = \emptyset$. So removing $W^l \cup W^r$ from bags of $\Omega^l - -P_2^c - -\Omega^r$ to obtain $\Omega^{cl} - -P_2^{cc} - -\Omega^{cr}$ does not cause any discontinuity of the subgraph induced in P'_1 by bags containing x, for any $x \in V$. Hence, P'_1 is a good path decomposition. Removing redundant bags yields a clique path, so H'_1 is an interval graph.

Claim 16. *uv is not an edge of H'_1.*

The graph H'_1 is constructed from three edge subsets of H_2, we have to show that the edge uv is not in any of them.

First, we prove that $uv \notin E(H_2[N_G[W^l] \cup S^l])$. We have $v \notin N_G[W^l]$. Indeed $v \notin N_G[W^l_{ini}]$ because all the components of $G - \Omega^r$ forming W^l_{ini} are contained in $\mathrm{V}(P^l - -P^l_1)$, and $v \notin N_G[W^l_{sup}]$ by construction of W^l_{sup}. Also $v \notin S^l$ because v is contained in the subpath $P_1^c - -K^l - -P_1^r - -P^r$ of P_1, and $v \notin K^r$.

Second, uv is not in $E(H_2[N_G[W^r] \cup S^r])$. Indeed $u \notin N_G[W^r]$ by construction of W^r_{sup} and by the fact that W^r_{ini} is a subset of $\mathrm{V}(P_1^r - -P^r)$. Clearly $u \notin S^r$ by Claim 10.

Third, since $u \in W^l$ the edge uv does not appear in $E(H_2 - (W^l \cup W^r))$.

We proved in Claim 16 that the edge uv does not exist in H'_1. Algorithm `TwoUnfolding` tries all possibilities for Ω^l, Ω^r, S^l, S^r, C^l, C^r and u and v. At some iteration it will make the good choice and construct the graph H'_1, an interval completion of G strictly contained in H_1. □

8.5 Conclusions

We give in this chapter a polynomial time algorithm verifying if a given interval completion is minimal. Given a non-minimal interval completion H of G, the algorithm computes a minimal one contained in H. Moreover, the process of removing edges which are not needed for the completion to be interval can be guided in order to obtain any minimal interval completion of G.

The natural continuation of this research aims in further characterizations of minimal interval completions. In would be very interesting to have a result defining a minimal interval completion H of an arbitrary graph G in terms of minimal separators of G. Such a characterization of minimal triangulations given in [99] proved useful in very interesting results on treewidth [25, 49].

Another direction of research to follow, is to work on technical details of the algorithm. With improved time complexity, it would be a very useful routine to use with heuristics for pathwidth. An efficient implementation could significantly increase the power of a heuristic for pathwidth or profile, by quickly removing unnecessary edges in the postprocessing.

Conclusions and Perspectives

The main motivation for this work comes from an important graph parameter of pathwidth. Encouraged by the successful attempts to exploit minimal chordal completions (triangulations) for computing treewidth, we decided to try the analogous approach for computing pathwidth; thus, to study minimal interval completions in the context of pathwidth. Despite the strong resemblances between these two families of problems based on embeddings into chordal (for the former) and interval (for the latter) graphs, very little was known on minimal interval completions. Moreover, virtually all results for pathwidth were based on treewidth.

As to our knowledge, we were the first to give a polynomial time algorithm computing minimal interval completions in [73], here presented in Chapter 6. It is an incremental algorithm, that can be used in the on-line fashion, computing a minimal interval completion each time a new vertex arrives, without modifying the completion computed in previous steps. The time complexity is of order $\mathcal{O}(n^3 \log n)$ ($\mathcal{O}(n^2 \log n)$ per vertex). The combinatorics of this algorithm is based on the characterization of interval graphs, as the ones that have clique paths [54]. The principal tool used here was the encoding of all possible clique paths of an interval graph by the pre-order relation, which we believe to be a promising tool for controlling path decompositions.

Then we discovered that the characterization of proper interval graphs as the ones that have bicompatible orderings [98] can be used for minimal proper interval completions. In [101], we gave a linear time algorithm computing a minimal proper interval completion, described in Chapter 3. Again, it was the first polynomial time algorithm for solving this problem ever published. It is a BFS algorithm with a simple tie-break rule that can be computed locally.

Working on [101] gave us better understanding of graph layouts and some tools very useful for controlling them. The intuitions proved to be true also for minimal interval completions, as we managed to extend our observations to the class of interval graphs. With this support, we gave an algorithm computing minimal interval completion [110], presented in Chapter 4, based on the characterization of interval graphs by interval orderings [97]. Like for proper interval completions, it is a BFS algorithm. Only that this time the tie-break rule is more involved, which increases the time complexity to $\mathcal{O}(nm)$. It is an open question, if the tie-break rule can be simplified. In particular, it is an interesting question if such a rule can be computed locally.

We also gave a polynomial time checkable characterization of minimal interval completions. In [72], we gave a polynomial time algorithm that verifies if a given interval completion H of G is minimal, described in Chapter 8. If it is not, the algorithm computes a minimal interval completion H' of G contained in H. The combinatorics here is based on foldings, a structural description of interval completions with respect to minimal interval completions that they contain. The feature of removing unnecessary fill edges from an arbitrary interval completion makes this algorithm especially interesting in conjunction with heuristics for pathwidth or profile, which usually do not yield a minimal interval completion. But there is some more work needed in order to reduce the

time complexity.

Finally, our studies on minimal interval completions have started to give fruits in the field of pathwidth. In [111], we give the first polynomial time algorithm computing pathwidth for a class of graphs of unbounded treewidth. Thanks to a generalization of tools used in [72], we characterized a particular set of interval completions of circular-arc graphs that contains solutions optimal for pathwidth. Based on this characterization, we give a $\mathcal{O}(n^2)$ time algorithm that computes the pathwidth of a circular-arc graph.

Having such a palette of tools controlling interval completions at hand it is only natural to ask for a domain where they could be of much help. Let us present a direction that we find promising.

As mentioned in the introduction, pathwidth is related to graph searching games. In particular, the pathwidth of G is equal the minimum number of searchers (minus one) needed to search G in the *invisible fugitive search game*. This game, first introduced by Kirousis et al.[81], is a one player game on an undirected graph G, using pebbles called searchers or guards. A *searching strategy* S is a sequence of moves where the player either places a guard on a vertex or removes a guard from a vertex. After each move, the status of vertices is evaluated. Initially all vertices are *unsafe*, that is to say, they may carry the fugitive. A vertex becomes *safe*, we know it does not carry the fugitive, when a searcher is put on it. A vertex becomes unsafe if it is connected by an unguarded path to an unsafe vertex, so it is possible that the fugitive takes this way. The purpose is to catch the fugitive, thus to make the whole graph safe. The goal of the game is to find a strategy that minimizes the number of searchers used. In the *visible fugitive search* version, the searchers can see the fugitive thus they can orient their search towards the area where the fugitive tries to escape. The necessary number of searchers might be much smaller than in the invisible version. E.g., two searchers can search any tree in the visible version, while the minimum number of searchers for the invisible fugitive version can be logarithmic in the size of the tree. It is interesting to mention here that the minimum number of searchers needed to clear the graph in the visible fugitive search on G equals the treewidth of G (plus one).

The search problems have been intensively investigated in the last few years. Several versions of the fugitive search game appeared, motivated by applications in network management. In particular, Fomin et al. [46, 47, 43, 45, 44] studied different measures for the quality of a search strategy and their relations to other graph parameters. Fraigniaud et al. [12, 52, 46, 5, 6] studied fugitive search with an additional condition, that the safe area should be connected. This is of particular importance if the search is to be performed in a distributed manner, by a team of agents with limited resources. Then, the agents need to have safe channels through the network to communicate. Another restriction which has been introduced is the logarithmic bound (in the size of the network) on the resources that the agents can use. Despite the attention that it has gained, the invisible fugitive game (in all its versions) has resisted attempts to create polynomial time exact algorithms or approximations. Most results just state the NP-hardness of new versions, not saying much on the positive side. We believe that deep insight into interval completions will help to create polynomial time algorithms computing approximations or exact solutions restricted classes, like it was the case with minimal triangulations and the visible fugitive game (treewidth). The motivation is even stronger in view of the growing interest of the network management community in solutions based on mobile agents [102, 89, 10]. In particular, it would be interesting to work on algorithms presented in Chapter 3 and 4, which look very promising from the distributed computation point of view. Even if the problems are NP-hard in general case, the properties of real-life large interaction networks might make them solvable in practice. In particular, the results on the small world phe-

nomenon (small diameter, greedy poly-logarithmic routing, etc.), might help in creating polynomial time algorithms for distributed searching in real-life situations (see [39, 51, 2]).

Bibliography

[1] K. Aardal, C. van Hoesel, A. Koster, C. Mannino, A. Sassano, *Models and solution techniques for the frequency assignement problem.* 4OR, 1(4): 261–317, 2003.

[2] I. Abraham, C. Gavoille, D. Malkhi, *Compact Routing for Graphs Excluding a Fixed Minor.* Proceedings of DISC 2005, 3724: 442–456, 2005.

[3] S. Arnborg, D. G. Corneil, A. Proskurowski, *Complexity of finding embeddings in a k-tree.* SIAM J. on Algebraic and Discrete Methods, 8: 277–284, 1987.

[4] S. Arnborg, A. Proskurowski, *Linear time algorithms for NP-hard problems restricted to partial k-trees.* Discrete Applied Mathematics, 23: 11–24, 1989.

[5] L. Barrire, P. Fraigniaud, N. Santoro, D. M. Thilikos, *Searching is not jumping.* Proceedings of WG 2003, LNCS 2880:34–45, 2003.

[6] L. Barrire, P. Flocchini, P. Fraigniaud, N. Santoro, *Connected Treewidth and Connected Graph Searching.* Proceedings of SPAA 2002, 200–209, ACM Press, 2002.

[7] A. Berry, J. P. Bordat, *Separability Generalizes Dirac's Theorem.* Discrete Applied Mathematics, 84(1-3): 43–53, 1998.

[8] A. Berry, J. P. Bordat, *Local LexBFS Properties in an Arbitrary Graph.* Proceedings of Journes Informatiques Messines, 2000.

[9] A. Berry, P. Heggernes, and Y. Villanger, *A vertex incremental approach for dynamically maintaining chordal graphs.* Proceedings of ISAAC 2003, LNCS, 2906:47–57, 2003.

[10] A. Bieszczad, B. Pagurek, T. White, *Mobile Agents for Network Management.* IEEE Communications Surveys and Tutorials, 1998.

[11] J. R. S. Blair and B. Peyton, *An introduction to chordal graphs and clique trees.* Graph Theory and Sparse Matrix Computations, 1–29, Springer, 1993.

[12] L. Blin, P. Fraigniaud, N. Nisse, D. M. Vial, *Distributed Chasing of Network Intruders.* Proceedings of SIROCCO 2006, LNCS 4056:70–84, 2006.

[13] H. L. Bodlaender, *Discovering Treewidth.* Proceedings of SOFSEM 2005, LNCS 3381:1–16, 2005.

[14] H. L. Bodlaender, *A tourist guide through treewidth.* Acta Cybernetica, 11: 1–23, 1993.

[15] H. L. Bodlaender, *A Linear-Time Algorithm for Finding Tree-Decompositions of Small Treewidth.* SIAM Journal on Computing, 25(6):1305–1317, 1996.

[16] H. Bodlaender, *A Partial k-Arboretum of Graphs with Bounded Treewidth.* Theoretical Computer Science, 209(1-2): 1–45, 1998.

[17] H. Bodlaender, F. Fomin, *Approximating the pathwidth of outerplanar graphs.* Journal of Algorithms, 43(2): 190–200, 2002.

[18] H. Bodlaender, T. Kloks, *Efficient and constructive algorithms for the pathwidth and treewidth of graphs.* Journal of Algorithms, 21(2): 358–402, 1996.

[19] H. L. Bodlaender, T. Kloks, D. Kratsch, *Treewidth and Pathwidth of Permutation Graphs.* SIAM J. Discrete Math., 8(4): 606–616, 1995.

[20] H. L. Bodlaender and A. Koster, *Safe separators for treewidth.* Technical Report UU-CS-2003-027, Institute of information and computing sciences, Utrecht University, Netherlands, 2003.

[21] H. L. Bodlaender, D. Thilikos, *Treewidth for graphs with small chordality.* Discrete Applied Mathematics, 79(1-3): 45–61, 1997.

[22] J.C. Boland, C.G. Lekkerkerker, *Representation of a finite graph by a set of intervals on the real line.* Fundamenta Mathematicae, 51:45–64, 1962.

[23] K. Booth and G. Leuker, *Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms.* J. Comput. Syst. Sci., 13:335–379, 1976

[24] V. Bouchitt and I. Todinca, *Approximating the treewidth of AT-free graphs.* Discrete Applied Mathematics, 131(1): 11–37, 2003.

[25] V. Bouchitté and I. Todinca, *Treewidth and minimum fill-in: Grouping the minimal separators.* SIAM J. Comput., 31:212–232, 2001.

[26] H. Broersma, E. Dahlhaus, T. Kloks, *Algorithms for treewidth and minimum fill-in of HHD-free graphs.* Proceedings of WG 1997, LNCS 1335: 109–117, 1997.

[27] H. Broersma, E. Dahlhaus, T. Kloks, *A linear time algorithm for minimum fill-in and treewidth for distance hereditary graphs.* Discrete Applied Mathematics, 99(1-3): 367–400, 2000.

[28] L. Cai, *Fixed-Parameter Tractability of Graph Modification Problems for Hereditary Properties.* Information Processing Letters, 58(4):171–176, 1996.

[29] P. Z. Chinn, J. Chvátalová, A. K. Dewdney, N. E. Gibbs, *The bandwidth problem for graphs and matrices - a survey.* Journal of Graph Theory, 6: 223–254, 1982.

[30] D. G. Corneil, *Lexicographic Breadth First Search - A Survey.* Proceedings of the 30th International Workshop on Graph-Theoretic Concepts in Computer Science, Lecture Notes in Computer Science, 3353: 1–19, 2004.

[31] B. Courcelle, *THe monadic second-order logic on graphs III: Treewidth, forbidden minors and complexity issues.* Informatique Théorique, 26: 257–286, 1992.

[32] B. Courcelle, J. Engelfriet, G. Rozenberg, *Contex-free handle-rewriting hypergraph grammars.* Graph-Grammars and Their Application to Computer Science, LNCS 532: 253–268, 1991.

[33] B. Courcelle, M. Mosbah, *Monadic second-order evaluations on tree-decomposable graphs.* Theoretical Computer Science, 109: 49-82, 1993.

[34] B. Courcelle, S. Olariu, *Upper bounds to the clique-width of graphs.* Discrete Applied Mathematics, 101: 77–114, 2000.

[35] A. Cournier and M. Habib, *A New Linear Algorithm for Modular Decomposition.* Proceedings of Trees in Algebra and Programming - CAAP'94, LNCS 787: 64–84, 1994

[36] J. Díaz, J. Petit, and M. J. Serna, *A survey of graph layout problems.* ACM Computing Surveys, 34:313–356, 2002.

[37] R. P. Dilworth, *A decomposition theorem for partially ordered sets.* Annals of Mathematics, 51:161–166, 1950.

[38] G. A. Dirac, *On Rigid Circuit Graphs.* Abh. Math. Sem. Univ. Hamburg, 21:71–76, 1961.

[39] P. Duchon, N. Hanusse, E. Lebhar, N. Schabanel, *Could any graph be turned into a small-world?* Theor. Comput. Sci., 355(1): 96–103, 2006.

[40] J. A. Ellis, M. Markov, *Computing the vertex separation of unicyclic graphs.* Information and Computation, 192(2): 123–161, 2004.

[41] J. A. Ellis, I. H. Sudborough, J. S. Turner, *The Vertex Separation and Search Number of a Graph.* Information and Computation, 113(1): 50–79, 1994.

[42] S. P. Fekete and J. Schepers, *A combinatorial characterization of higher-dimensional orthogonal packing.* Mathematics of Operations Research, 29:353–368, 2004.

[43] F. V. Fomin, *Searching expenditure and interval graphs.* Discrete Applied Mathematics, 135(1-3): 97–104, 2004.

[44] F. V. Fomin, *Helicopter Search Problems, Bandwidth and Pathwidth.* Discrete Applied Mathematics, 85(1): 59–70, 1998.

[45] F. V. Fomin, P. A. Golovach, *Graph Searching and Interval Completion.* SIAM J. Discrete Math., 13(4): 454–464, 2000.

[46] F. V. Fomin, P. Fraigniaud, N. Nisse, *Nondeterministic Graph Searching: From Pathwidth to Treewidth.* Proceedings of MFCS 2005, LNCS 3618: 364–375, 2005.

[47] F. V. Fomin, P. Heggernes, J. A. Telle, *Graph Searching, Elimination Trees, and a Generalization of Bandwidth.* Algorithmica, 41(2): 73–87, 2004.

[48] F. V. Fomin, D. Kratsch, H. Mller, *On the Domination Search Number.* Discrete Applied Mathematics, 127(3): 565–580, 2003.

[49] F. V. Fomin, D. Kratsch, and I. Todinca, *Exact (exponential) algorithms for treewidth and minimum fill-in.* Proceedings of ICALP 2004, LNCS, 3142: 568–580, 2004.

[50] F. Fomin, D. Thilikos, *A 3-approximation for the pathwidth of Halin graphs.* To appear in Journal of Discrete Algorithms.

[51] P. Fraigniaud, *Greedy Routing in Tree-Decomposed Graphs.* Proceedings of ESA 2005, LNCS 3669: 791–802, 2005.

[52] P. Fraigniaud, N. Nisse, *Connected Treewidth and Connected Graph Searching.* Proceedings of LATIN 2006, LNCS 3887: 479–490, 2006.

[53] M. K. Franklin, Z. Galil, M. Yung, *Eavesdropping games: a graph-theoretic approach to privacy in distributed systems.* Journal of ACM, 47(2): 225–243, 2000.

[54] D. R. Fulkerson, O. A. Gross *Incidence matrices and interval graphs.* Pacific journal of mathematics, 15(3): 835–855, 1965.

[55] T. Gallai, *Transitiv orienterbare graphe.* Acta Math. Acad. Sci. Hungar, 18: 25–66, 1967.

[56] M. R. Garey and D. S. Johnson, *Computer and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[57] F. Gavril, *The intersection graphs of subtrees in trees are exactly the chordal graphs.* Journal of Combinatorial Theory B, 16: 47–56, 1974.

[58] J. A. George and J. W. H. Liu, *Computer Solution of Large Sparse Positive Definite Systems.* Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1981.

[59] P. C. Gilmore and A. J. Hoffman, *A characterization of comparability graphs and of interval graphs.* Canadian Journal of Mathematics, 16: 539–548, 1964.

[60] P. W. Goldberg, M. C. Golumbic, H. Kaplan, and R. Shamir, *Four strikes against physical mapping of DNA.* Journal of Computational Biology, 2(1): 139–152, 1995.

[61] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs.* Academic Press, San Diego, 1984.

[62] G. Gottlob, M. Grohe, N. Musliu, M. Samer, F. Scarcello, *Hypertree decompositions: Structure, algorithms, and applications.* Proceedings of WG 2005, LNCS 3787: 1-15, 2005.

[63] J. Gustedt, *On the pathwidth of chordal graphs.* Discrete Applied Mathematics, 45(3): 233–248, 2003.

[64] G. Gutin, S. Szeider, and A. Yeo, *Fixed-Parameter Complexity of Minimum Profile Problems.* Proceedings of IWPEC 2006, LNCS, 4169, 2006. To appear.

[65] M. Habib, R. M. McConnell, C. Paul, L. Viennot, *Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing.* Theoretical Computer Science, 234(1-2): 59–84, 2000.

[66] M. Habib, R. Moehring, *Treewidth of cocomparability graphs and a new order-theoretic parameter.* ORDER, 1: 47–60, 1994.

[67] M. Habib, C. Paul, L. Viennot, *Partition Refinement Techniques: An Interesting Algorithmic Tool Kit.* International Journal of Foundations of Computer Science, 10(2): 147–170, 1999.

[68] T. Hagerup, *Dynamic algorithms for graphs of bounded treewidth.* Proceedings of ICALP 1997, LNCS, 292–302, 1997.

[69] P. Heggernes, *Minimal triangulations of graphs: A survey.* Discrete Mathematics, 306(3): 297–317, 2006.

[70] P. Heggernes, F. Mancini, *Minimal Split Completions of Graphs.* Proceedings of LATIN 2006, Lecture Notes in Computer Science, 3887: 592–604, 2006.

[71] P. Heggernes, F. Mancini, C. Papadopoulos, *Minimal Comparability Completions.* Tech. Report, University of Bergen, 2006, http://www.ii.uib.no/publikasjoner/texrap/pdf/2006-317.pdf

[72] P. Heggernes, K. Suchan, I. Todinca, and Y. Villanger, *Characterizing minimal interval completions: Towards better understanding of profile and pathwidth.* Research Report RR-2006-09, LIFO - University of Orléans,2006.

[73] P. Heggernes, K. Suchan, I. Todinca,Y. Villanger, *Minimal Interval Completions.* Proceedings of ESA 2005, LNCS 3669: 403–414, 2005.

[74] P. Heggernes, J. A. Telle, Y. Villanger, *Computing minimal triangulations in time $O(n^{\alpha}logn) = o(n^{2.376})$.* Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms - SODA 2005, SIAM, 907–916, 2005.

[75] C. van Hoesel, A. Koster, A. Kolen, *Optimal solutions for a frequency assignement problem via tree-decomposition.* Proceedings of WG 1999 LNCS 1665: 338–349, 1999.

[76] L. Ibarra, *The clique-separator graph for chordal graphs and subclasses of chordal graphs.* Presented at Symposium on Discrete Mathematics, Nashville, TN, 2004.

[77] P. Jégou, S. Ndiaye, C. Terrioux, *Computing and exploiting tree-decompositions for solving constraint networks.* Proceedings of CP 1995, 777–781, 2005.

[78] P. Jégou, C. Terrioux, *Hybrid backtracking bounded by tree-decomposition of constraint networks.* Artificial Intelligence, 146(1): 43–75, 2003.

[79] H. Kaplan, R. Shamir, *Pathwidth, Bandwidth, and Completion Problems to Proper Interval Graphs with Small Cliques.* SIAM Journal on Computing, 25(3): 540–561, 1996.

[80] H. Kaplan, R. Shamir, R. E. Tarjan, *Tractability of Parameterized Completion Problems on Chordal, Strongly Chordal, and Proper Interval Graphs.* SIAM Journal on Computing, 28(5): 1906–1922, 1999.

[81] M. Kiriousis, C. Papadimitriou, *Searching and pebbling.* Theor. Comput. Sci., 42(2): 205–218, 1986.

[82] D. J. Kleitman, R. Vohra, *Computing the Bandwidth of Interval Graphs.* SIAM J. Discrete Math., 3(3): 373–375, 1990.

[83] T. Kloks, *Treewidth of circle graphs.* Intern. J. Foun. Comput. Sci., 7: 111–120, 1996.

[84] T. Kloks, D. Kratsch, *Treewidth of chordal bipartite graphs.* Journal of Algorithms, 19(2): 266–281, 1995.

[85] T. Kloks, D. Kratsch, and J. Spinrad, *On treewidth and minimum fill-in of asteroidal triple-free graphs.* Theoretical Computer Science, 175: 309–335, 1997.

[86] T. Kloks, D. Kratsch, C. K. Wong, *Minimum Fill-in on Circle and Circular-Arc Graphs.* J. Algorithms 28(2): 272–289, 1998.

[87] D. Kratsch and J. P. Spinrad, *Between $O(nm)$ and $O(n^\alpha)$.* Proceedings of SODA 2003, 709–716, 2003.

[88] D. Kratsch, J. Spinrad, *Minimal fill in $\mathcal{O}(n^{2.69})$ time.* Discrete Mathematics, 306(3): 366–371, 2006.

[89] A. Liotta, G. Pavlou, and G. Knight, *Exploiting Agent Mobility for Large Scale Network Monitoring.* IEEE Network, 16(3): 7–15, 2002.

[90] R.M. McConnell, *Linear-Time Recognition of Circular-Arc Graphs.* Algorithmica, 37(2): 93–147, 2003.

[91] N. Meggido, S. L. Hakimi, M.R. Garey, D.S. Johson, C.H. Papadimitriou, *The complexity of searching a graph.* Journal of the ACM, 35: 18–44, 1988.

[92] T. A. McKee, F. R. McMorris, *Intersection graph theory.* SIAM, 1999.

[93] R. H. Mohring, D. Wagner, F. Wagner, *VLSI network design, a survey.* Handbooks in Operations Research/Management Science, Volume on Networks, 625–712, 1995.

[94] B. Monien, *The bandwidth minimization problem for caterpillars with hair length 3 in NP-complete.* SIAM Journal on Algebraic and Discrete Methods, 7: 505–512, 1986.

[95] A. Natanzon, R. Shamir, and R. Sharan, *A polynomial approximation algorithm for the minimum fill-in problem.* SIAM Journal on Computing, 30(4): 1067–1079, 2000.

[96] A. Natanzon, R. Shamir, and R. Sharan, *Complexity classification of some edge modification problems.* Discrete Applied Mathematics, 113: 109–128, 2001.

[97] S. Olariu, *An optimal greedy heuristic to color interval graphs.* Information Processing Letters, 37(1): 21–25, 1991.

[98] B. S. Panda, S. K. Das, *A linear time recognition algorithm for proper interval graphs.* Information Processing Letters, 87(3): 153–161, 2003.

[99] A. Parra and P. Scheffler, *Characterizations and algorithmic applications of chordal graph embeddings.* Discrete Applied Mathematics, 79: 171–188, 1997.

[100] B. W. Peyton, *Minimal orderings revisited.* SIAM J. Matrix Anal. Appl., 23(1): 271–294, 2001.

[101] I. Rappaport, K. Suchan, I. Todinca, *Minimal proper interval completions.* Proceedings of WG 2006, LNCS, 2006. To appear.

[102] S. Rayan, R. Pradeep, N. Paramesh, *Network management platform based on mobile agents.* International Journal of Network Management, 14: 59–73, 2004.

[103] N. Robertson, P. D. Seymour, *Graph minors I. Excluding a forest.* Journal of Combinatorial Theory. Series B, 35: 39–61, 1983.

[104] N. Robertson, P. D. Seymour, *Graphs minors II. Algorithmic aspects of tree-width.* Journal of Algorithms, 7: 309–322, 1986.

[105] N. Robertson, P. D. Seymour, *Graph minors III. Planar tree-width.* Journal of Combinatorial Theory. Series B, 36: 49–64, 1984.

[106] D. J. Rose, *On simple characterization of k-trees.* Discrete Mathematics, 7: 317–322, 1974.

[107] D. Rose, R.E. Tarjan, and G. Lueker, *Algorithmic aspects of vertex elimination on graphs.* SIAM Journal on Computing, 5: 146–160, 1976.

[108] P. Seymour, R. Thomas, *Graph searching, and a min-max theorem for tree-width.* J. of Combinatorial Theory, 58(1): 22–33, 1993.

[109] K. Skodinis, *Construction of linear tree-layouts which are optimal with respect to vertex separation in linear time.* J. Algorithms, 47(1): 40–59, 2003.

[110] K. Suchan, I. Todinca, *Minimal interval completion through graph exploration.* Proceedings of ISAAC 2006, LNCS, 2006. To appear.

[111] K. Suchan, I. Todinca, *Pathwidth of circular-arc graphs.* Research Report RR-2006-10, LIFO - University of Orléans, 2006.

[112] R. Sundaram, K. Sher Singh, C. Pandu Rangan, *Treewidth of circular-arc graphs.* SIAM Journal on Discrete Mathematics, 7: 647–655, 1994.

[113] E. Szpilrajn-Marczewski, *Sur deux propriétés des classes d'ensembles.* Fundamenta Mathematicae, 33: 303-307, 1945.

[114] E. Wanke, *k-NLC graphs and polynomial algorithms.* Discrete Applied Mathematics, 54(2-3): 251–266, 1994.

[115] M. Yannakakis, *Edge-deletion problems.* SIAM Journal on Computing, 10(2): 310–327, 1981.

[116] M. Yannakakis, *Computing the minimum fill-in is NP-complete.* SIAM J. Alg. Disc. Meth., 2: 77–79, 1981.

Complétions d'intervalles minimales

Résumé : La *largeur linéaire* et la *largeur arborescente* ont été introduites par Robertson et Seymour dans leurs travaux sur les mineurs de graphes. De manière informelle, la largeur linéaire (resp. la largeur arborescente) d'un graphe mesure l'écart entre ce graphe et la classe des chaînes (des arbres). Les deux paramètres se sont révélés très puissants de point de vue algorithmique, car de nombreux problèmes NP-difficiles deviennent polynomiaux lorsque l'on se restreint à des classes de graphes de largeur linéaire ou de largeur arborescente bornée. Pour les graphes de petite largeur linéaire, les algorithmes de programmation dynamique utilisés pour la résolution des problèmes difficiles ont une complexité en espace bien meilleure que pour les graphes de petite largeur arborescente.

Etant donné un graphe G=(V,E) quelconque, un graphe d'intervalles H=(V,F) tel que G soit un sous-graphe de H est appelé *complétion d'intervalles* de G. La largeur linéaire de G est le minimum de ω(H)-1, parmi toutes les complétions d'intervalles H de G ; ω(H) désigne la taille de la clique de cardinal maximum de H. Calculer une complétion d'intervalles qui réalise ce minimum est NP-difficile. C'est pourquoi nous calculerons des *complétions d'intervalles minimales*, où l'on demande seulement que l'ensemble d'arêtes rajoutées F\E soit minimal par inclusion parmi toutes les complétions possibles. Parmi ces complétions d'intervalles minimales on trouve les complétions optimales pour la largeur linéaire. Une approche similaire, à travers les triangulations minimales, est fortement utilisée pour comprendre et calculer la largeur arborescente.

Ce mémoire présente nos résultats sur les complétions d'intervalles minimales. Nous donnons trois algorithmes calculant une complétion d'intervalles minimale, basés sur des approches différentes. Nous présentons également un algorithme calculant une complétion d'intervalles propres minimale. Enfin, nous montrons que la largeur linéaire des graphes d'intervalles circulaires peut être calculée en temps polynomial.

Mots-clés : graphe, algorithme, décomposition linéaire, graphe d'intervalle, completion d'intervalles minimale

Minimal Interval Completions

Abstract : *Pathwidth* and *treewidth* were introduced by Robertson and Seymour in their work on graph minors. Informally, the pathwidth (resp. treewidth) of a graph measures the distance between this graph and the class of paths (trees). Both parameters proved algorithmically very useful, as many classical NP-hard problems are polynomial time tractable on graphs of bounded pathwidth or treewidth. For graphs of small pathwidth the dynamic programming algorithms used for solving hard problems have a better space complexity than in case of small treewidth.

An interval supergraph H=(V,F) of G=(V,E) is an *interval completion* of G. The pathwidth of G is defined as the minimum cliquesize of H, over all interval completions H of G, minus one. Finding an interval completion of minimum cliquesize (pathwidth) is NP-hard. In *minimal interval completions*, one only asks the set F\E of added edges to be inclusion minimal. Among them, one finds pathwidth optimal completions. A similar approach, through minimal chordal completions, proved useful for tackling treewidth. Thus minimal interval completions should help understand pathwidth. The report gives our results on interval completions. There are three polynomial time algorithms computing minimal interval completions, based on different aspects of the problem. We also present a linear time algorithm computing minimal proper interval completions. Finally, there is a polynomial time algorithm computing the pathwidth of circular-arc graphs.

Key words: graph, algorithm, path decomposition, interval graph, minimal interval completion

DISCIPLINE - SPECIALITE DOCTORALE : Informatique

LABORATOIRE : LIFO, Université d'Orléans, B.P. 6759, F-45067 ORLEANS Cedex 2