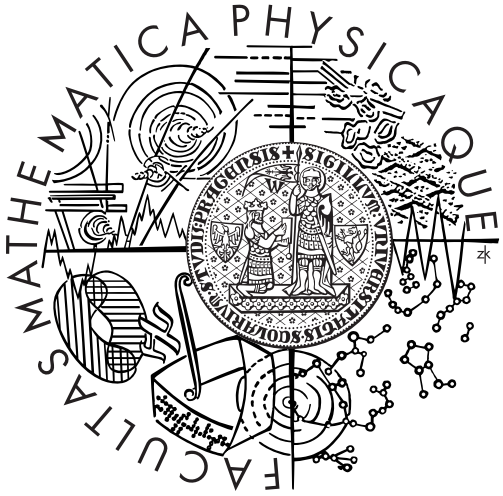
Charles University in Prague
Faculty of Mathematics and Physics

Bachelor's Thesis



Pavel Klavík

Geometric Representations of Graphs

Department of Applied Mathematics

Supervisor

Prof. RNDr. Jan Kratochvíl, CSc.

Study program

General Computer Science

2010

Acknowledgement

My special thanks belong to my advisor, Prof. Jan Kratochvíl, for leading me through my studies and research.

I am grateful to Tomáš Vyskočil for fruitful discussions concerning results presented in this thesis.

Also, I am grateful to Jiří Fink for the LaTeX style used in this work which is based on his doctoral thesis. I would like to thank Martin Mareš for valuable remarks concerning typesetting.

Last but not least, I would like to thank all the members and the students of the Department of Applied Mathematics for the wonderful environment, support, and excellent background without which I could hardly write this thesis.

Hereby I declare that I have written this thesis on my own, and the references include all the sources of information I have exploited. I agree with the lending of this thesis.

Prague, August 5, 2010 Pavel Klavík

Contents

Název práce:	Geometrické representace grafů
Autor:	Pavel Klavík
Katedra:	Katedra aplikované matematiky
Vedoucí práce:	Prof. RNDr. Jan Kratochvíl, CSc.
E-mail vedoucího:	honza@kam.mff.cuni.cz
Klíčová slova:	průnikové grafy, rozpoznávání, složitost, algoritmy

Abstrakt:

Průnikové grafy jsou podrobně studovaným odvětvím teorie grafů. Složitostní otázky rozpoznávání jsou zkoumány již řadu let. Pro zadaný graf se ptáme, zda patří do dané třídy. V této práci představujeme nový problém rozšiřování částečných reprezentací. Pro tento problém je vedle grafu zafixovaná část reprezentace. Ptáme se, zda je možné tuto částečnou reprezentaci rozšířit na celý graf. Tento problém je alespoň tak težký jako rozpoznávání.

Zabýváme se problémem rozšiřování pro několik tříd průnikových grafů. Vyřešili jsme rozšiřování intervalových grafů v čase $\mathcal{O}(n^2)$ a vlastních intervalových grafů v čase $\mathcal{O}(mn)$. Pomocí metody popsané Golumbicem umíme rozšířit porovnatelné a permutační grafy v čase $\mathcal{O}(\Delta \cdot m)$.

Mezi některými třídami jsou známy rovnosti, například mezi jednotkovými intervalovými grafy a vlastními intervalovými grafy. Překvapivě v případě rozšiřovaní je musíme rozlišit. Podobně ukazujeme, že v případě rozšiřovaní funkčních grafů a doplňků porovnatelných grafů se jedná o zcela jiné problémy.

Title:	Geometric Representations of Graphs
Author:	Pavel Klavík
Department:	Department of Applied Mathematics
Supervisor:	Prof. RNDr. Jan Kratochvíl, CSc.
Supervisor's e-mail:	honza@kam.mff.cuni.cz
Keywords:	intersection graphs, recognition, complexity, algorithms

Abstract:

Intersection graphs are a well studied field of graph theory. Complexity questions of recognition have been studied for several years. Given a graph, we ask whether the graph belongs to a fixed class. In this thesis, we introduce a new problem of partial representation extension. In this problem, aside from a graph, a part of a representation is also fixed. We ask whether it is possible to extend this partial representation to the whole graph. This problem is at least as hard as recognition.

We study the partial representation extension problem for several intersection defined classes. We solve extending of interval graphs in time $\mathcal{O}(n^2)$ and proper interval graphs in time $\mathcal{O}(mn)$. Using an approach described by Golumbic, we further show that comparability and permutation graphs are extendable in time $\mathcal{O}(\Delta \cdot m)$.

There are some classes that are known to be equal, for example unit interval graphs and proper interval graphs. Surprisingly, in the case of extending, we need to distinguish them. Similarly, we show that extending of function graphs and extending of co-comparability graphs are completely different problems.

Introduction

Studies of representations of graphs are as old as graph theory. An example of a representation is a drawing of graphs in the plane. Graphs that admit drawing without edge crossings, called planar graphs, were studied already by Euler. Kuratowski [Kur30] gave their first combinatorial characterization which can be considered as the start of modern graph theory.

Recognition. For a class, the computational problem of recognition is considered. Given a graph, we ask whether the graph belongs to this class. The result of Kuratowski implies that planar graphs can be recognized in finite time, using just their combinatorial structure. Nowadays, there several algorithms known for recognition of planar graphs in linear time, see [HT74, BL76, BM04].

Partial Representation Extension. In this thesis, we introduce a similar problem called *partial representation extension*. Given a graph with a part of the representation fixed, we ask whether it is possible to extend this representation to the whole graph. For example, we get as an input a planar graph with some vertices and edges already placed in the plane. The problem asks whether the rest of the graph can be drawn to obtain a planar representation. For the case of planar graphs, this problem is considered in a recent paper of Kratochvíl et al. [ABF$^+$10] and a linear time algorithm for planar graph extension is given.

Every planar graph admits an embedding using straight-line edges, moreover we can find it in linear time [Sch90]. Surprisingly, the problem of partial representation extension for a representation using only straight-line edges is an NP-complete problem [Pat06].

Intersection graphs. An intersection graph has a representation that assigns a set to every vertex in such a way that two vertices are adjacent if and only if the corresponding sets intersect. Marczewski [Mar45] showed that every undirected graph can be represented as an intersection graph. In theory and applications, we consider classes of intersection graphs having all the assigned sets of some specific properties—for example, geometric or combinatorial. We present several examples of these classes.

Examples of Classes. *String graphs* are intersection graphs of continuous curves, called strings, in the plane. They have several interesting properties, for example, every planar graph is a string graph. Their recognition was proven to be NP-hard by Kratochvíl [Kra91], but it was open for a long time whether it belongs to NP. Recently,

it was proved that they belong to NP [SSŠ02]. String graphs have several important subclasses.

Segment graphs are intersection graphs of segments in the plane. It is interesting that their recognition is known to be NP-hard [Kra91], but it is open whether it belongs to NP.

Interval graphs are intersection graphs of closed intervals of the real line. This class was intensively studied and it has several interesting properties. Recognition of interval graphs lies in P, several linear algorithms are known. We describe them in more detail in Chapter 2.

Books. Intersection graphs are a well-studied field of graph theory. Throughout this thesis, we expect the reader is familiar with their basics. Key properties are described but proofs are mostly omitted or written in sketches. There are several great books describing intersection graphs, for example [Gol04, Spi03, MM99]. If the reader is not familiar with the basics of computational complexity, we recommend excellent books [Pap94, AB09].

Finding all the Representations. We present one more property of a recognition algorithm which is desirable. In certain applications, it is useful to find all the possible representations, or at least enumerate a few of them.

We give an example of such an application, described in [MPT98]. In molecular biology, there is a method called hybridization used to get a physical map of a long DNA molecule. A scientist extracts several scratches, called *clones*, and analyzes them. The problem is how to place them correctly along the molecule. Several unique positions in DNA called *probes* are given. We only know the results of tests that tell us which clones contain which probes. Using this data, we want to reconstruct the positions of the clones.

Since DNA has a linear structure, every clone corresponds to an interval and every probe is a fixed point of the real line. We want to find a representation in which every clone intersects the correct probes. It is very useful to know all the possible representations. The scientist can verify whether the data is enough to produce a unique solution. In the case there exist more solutions, other probes can be introduced. Greenberg and Istrail [GI95] describe more details about this approach.

In other applications, we want to find a representation having some properties. We know that these properties are very likely. We can enumerate different representations until we find one having the desired properties.

Notice that we cannot expect to find all the possible representations in polynomial time—a graph can admit even an exponential number of topologically different representations. Therefore, we measure the time we need to find the next representation—we call this time a *delay*. It is desired to find an algorithm with a polynomial delay. All the algorithms described in this thesis can find all the representations with a polynomial delay, since we modify recognition algorithms having this property.

Our Approach. In this thesis, we consider complexity of partial representation extension for several classes of intersection graphs. Our main concern is to determine whether the problems are solvable in polynomial time, or are NP-complete (unless P = NP). It is quite likely that the algorithms described in this thesis are not optimal.

Also, the reader may notice that the problem of partial representation extension is at least as hard as the corresponding recognition problem—the partial representation does not have to fix anything. Therefore, the classes for which recognition is NP-complete and trivially in NP (unlike string graphs) are not interesting from this point of view. Somewhat surprisingly, all the classes solved in this thesis turned out to be extendible in polynomial time.

The Structure. This thesis has the following structure. In the next chapter, we define the problems formally. The other chapters describe how to solve partial representation extension for specific classes.

In Chapter 2, we give an $\mathcal{O}(n^2)$ algorithm for extending interval graphs.

In Chapter 3, we consider two subclasses of interval graphs—proper interval graphs and unit interval graphs. A classic result states that they are equivalent. Surprisingly, the problem of partial representation extension distinguishes them. We modify the algorithm from Chapter 2 to extend proper interval graphs in time $\mathcal{O}(mn)$. The complexity of extending unit interval graphs remains open.

In Chapter 4, we consider three classes—comparability graphs, permutations graphs and function graphs. Similarly to Golumbic [GS93], we solve extension of comparability graphs in time $\mathcal{O}(\Delta \cdot m)$. The main result is that we can extend permutation graphs in time $\mathcal{O}(\Delta \cdot m)$. The complexity of extending function graphs remains open. Similarly to unit interval graphs, for the problem of partial representation extension of function graphs, we cannot use extending of an equivalent class of co-comparability graphs.

In the conclusion, we describe several related questions and state a list of open problems.

Chapter 1

Definitions

We start with basic notions and formal definitions. Also, we give precise definitions of the recognition and the partial representation extension problems. Definitions of specific classes of graphs are contained in the following chapters.

Graphs. A graph G is an ordered pair (V, E) of vertices V and edges E. If not stated otherwise, we consider only simple, finite and undirected graphs. For a graph G, we denote the number of the vertices by n, the number of the edges by m and the maximal degree by Δ.

Representations. Except comparability graphs in Chapter 4, we consider only intersection graphs and their representations. For a given intersection graph, a *representation* is an assignment of sets to the vertices such that two sets intersect if and only if the corresponding vertices are adjacent. Specific classes of intersection graphs admit only sets having some property. For example, interval graphs are represented by intervals of the real line.

Problem:	Recognition of $\mathcal{C}$ – $\textsc{Recog}(\mathcal{C})$.
Input:	A graph G.
Output:	A representation of G if G belongs to the class $\mathcal{C}$, "no" otherwise.

Partial Representations. A *partial representation* is a representation of an induced subgraph. A representation *extends* a partial representation if the sets assigned to the vertices of the subgraph are the same.

Problem:	Partial Representation Extension of $\mathcal{C}$ – $\textsc{PRExt}(\mathcal{C})$
Input:	A graph G with a partial representation R.
Output:	A representation of G extending R if it exists, "no" otherwise.

Chapter 2

Extending Interval Graphs

Interval graphs are intersection graphs of (closed) intervals of the real line, first considered by Hajós [Haj57]. We denote this class **INT**.

Motivations. Their research is motivated by many applications, for example, in biology, psychology and traffic light scheduling, see [Rob76, Sto68]. When the structure of DNA was not understood, Benzer [Ben59] used interval graphs to study the topology of DNA. We sketch his approach.

Benzer studied genes of a simple microorganism. The DNA molecule can appear in the original form or with a mutation. A mutation changes a consecutive part of the DNA. Benzer analysed 159 different mutations. For every pair of mutations, he measured whether their changes intersect. So, an intersection graph of the mutations was constructed. If DNA would have a linear topology, this graph would be an interval graph. Benzer found an interval representation of the mutations which gave a strong evidence that the topology of DNA is linear. Nowadays, everyone knows that topology of DNA is linear, forming a double helix, so this application has only a historical meaning.

Interval graphs are also interesting from the theoretical point of view. Their definition is very simple but they are non-trivial to characterize. On the other hand, they are recognizable in polynomial time. Moreover, several problems NP-complete for general graphs, as MaximumClique or k-Coloring, can be solved for interval graphs in polynomial time.

Our approach. In this section, we describe an algorithm solving PRExt(**INT**) in time $\mathcal{O}(n^2)$. We consider only the representations having the endpoints of all the intervals distinct since it simplifies the description. It is possible to generalize the algorithm for intervals sharing endpoints.

The algorithm can also find all the different representations with a polynomial delay $\mathcal{O}(n^2)$. Two representations are different if their orderings of the maximal cliques are different. The importance of maximal cliques is explained later.

Recognition. Recognition of interval graphs in linear time was a long-standing open

problem, first solved by Booth and Lueker [BL76] using PQ-trees. Nowadays, there are two main approaches to recognition in linear time. The first approach finds a feasible ordering of the maximal cliques which can be done using a data structure called PQ-tree. The second one uses surprising properties of the lexicographic breadth-first search (LBFS), searches through the graph several times and constructs a representation if the graph is an interval graph [COS09].

The algorithm based on LBFS is very simple to implement, but quite complicate to prove. On the other hand, PQ-trees are more difficult to implement, but the PQ-Tree Algorithm is very robust. It allows to find all the representations with a polynomial delay $\mathcal{O}(n)$. Also, it is useful for solving other problems, for example testing of planarity and consecutive one property of matrices. A more recent paper [MPT98] describes a slightly modified structure called PQR-trees. We modify PQ-trees to solve PREXT(**INT**) in time $\mathcal{O}(n^2)$.

PQ-trees. To explain the modification, we first describe how the PQ-Tree Algorithm works. It is based on the following characterization of interval graphs, due to Fulkerson and Gross [FG65]:

Lemma 2.1 ([FG65])**.** *A graph is an interval graph if and only if there exists an ordering of the maximal cliques such that for every vertex the cliques containing this vertex appear consecutively in this ordering.*

It is easy to see that this characterization holds. If a graph is an interval graph, there exists an interval representation, see Figure 2.1. Since the intervals have the Helly property, for every maximal clique there exists a point of the real line contained in all the intervals of this clique. These points are representing the maximal cliques. We call them *clique-points*. An ordering of the maximal cliques is given by the ordering of the clique-points on the real line from left to right. Notice that every vertex is contained in consecutive cliques since it is represented by an interval.

On the other hand, given an ordering of the maximal cliques, we place clique-points in this ordering on the real line, see Figure 2.1. For every vertex, we assign an interval containing exactly all the clique-points representing the cliques containing this vertex. Observe that we obtain a valid interval representation of the graph.

In the rest of the section, by a clique we mean a maximal clique. Every interval graph is also a chordal graph. For a chordal graph, all the cliques have in total $\mathcal{O}(n+m)$ vertices. Moreover, we can find all the cliques in linear time using a genius algorithm by Tarjan et al. [RTL76]. Therefore, we have reduced the problem of interval graph

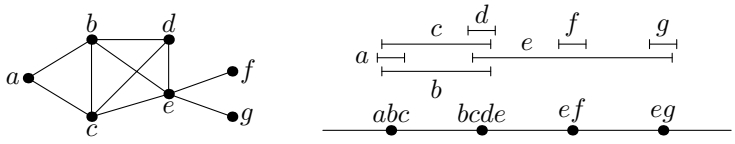


Figure 2.1: An interval graph and its representation with the corresponding clique-points.

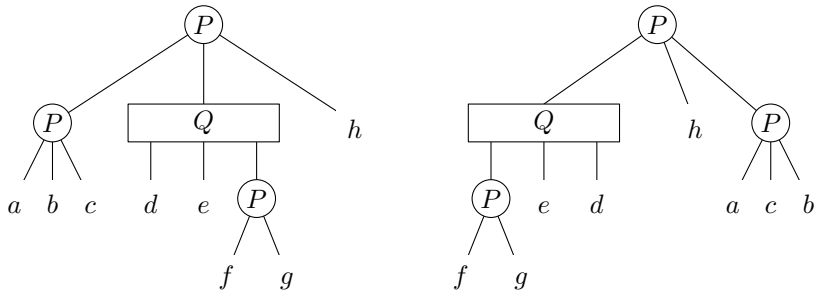


Figure 2.2: PQ-trees representing permutations $abcdefgh$ and $fgedhacb$.

recognition to finding a feasible ordering of the cliques. We solve this problem using PQ-trees.

We have the following problem: For given elements and their restricting subsets, we want to find a permutation of elements in which every set appears consecutively. A PQ-tree is a tree structure that stores all the feasible permutations. The leaves correspond one-to-one to the elements of the permutation. The inner nodes are of two types: the P-nodes and the Q-nodes. For every node, the order of its children is fixed. A PQ-tree represents one permutation, given by the ordering of leaves from left to right, see Figure 2.2.

To obtain other feasible permutations, trees admit two operations. The children of a P-node can be reordered in an arbitrary way. On the other hand, we can only reverse the order of the children of a Q-node. Two trees are equivalent if we can change one tree to the other one only using these operations. For example, the trees in Figure 2.2 are equivalent. Every equivalence class of the trees corresponds to all the permutations feasible for some input sets.

Booth and Lueker [BL76] describe how to construct a PQ-tree in linear time $\mathcal{O}(e + f + t)$, where e is the number of elements, f is the number of sets and t is the total size of all the sets. Therefore, we can recognize interval graphs in $\mathcal{O}(n + m)$.

Extending INT. We first sketch the algorithm for solving PREXT(**INT**). We construct a PQ-tree for the input graph, completely ignoring the given partial representation. We want to find a feasible ordering of the cliques. The partial representation gives another restriction—a partial ordering of the cliques. We prove that if there exists an ordering of the cliques compatible with the PQ-tree extending this partial ordering, the representation can always be extended. Otherwise, it is not possible to extend the partial representation.

For a given ordering of the cliques, we place clique-points on the real line. We need to be more careful in this step. Since several intervals are fixed, we cannot change their representations. Using the clique-points, we construct a representation in a similar manner as in Figure 2.1.

Now, we describe everything with all the details.

Clique Bounds $\curvearrowleft$ and $\curvearrowright$. For a clique a, let $I(a)$ denote the set of all the intervals

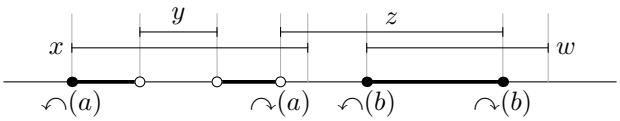


Figure 2.3: Clique-points a and b, having $I(a) = \{x\}$ and $I(b) = \{z, w\}$, can be placed to the bold parts of the real lines.

placed by the partial representation that are contained in a. We want to place the clique-point a to a part of the real line containing exactly all the intervals of $I(a)$ and no others, see Figure 2.3.

We denote $\curvearrowleft(a)$ (resp. $\curvearrowright(a)$) the leftmost (resp. the rightmost) point where the clique-point a can be placed, formally:

$$\begin{aligned} \curvearrowleft(a) &= \inf\big\{x \mid \text{the clique-point } a \text{ can be placed to } x\big\}, \\ \curvearrowright(a) &= \sup\big\{x \mid \text{the clique-point } a \text{ can be placed to } x\big\}. \end{aligned}$$

Notice that it does not mean that this clique-point a can be placed to all the points between $\curvearrowleft(a)$ and $\curvearrowright(a)$. If a has $\curvearrowleft(a) > \curvearrowright(a)$, the clique-point cannot be placed at all and the given partial representation is not extendible.

Clique Ordering $\blacktriangleleft$. The partial representation gives one more restriction. If two cliques a and b have $\curvearrowright(a) \leq \curvearrowleft(b)$, every correct ordering has to place a before b. So, the algorithm checks these conditions for all the cliques and obtains a partial ordering $\blacktriangleleft$. For example, the cliques a and b in Figure 2.3 satisfy $a \blacktriangleleft b$.

Steps of the Algorithm. The algorithm consists of the following steps:

1. We find maximal cliques and construct a PQ-tree, independently of the partial representation.
2. We compute $\curvearrowleft$ and $\curvearrowright$ for all the cliques and construct $\blacktriangleleft$.
3. We search the PQ-tree and find an ordering of the cliques extending $\blacktriangleleft$.
4. We place the clique-points on the real line.
5. Using the clique-points, we construct a representation.

Step 1: Constructing a PQ-tree. We can finish the first step in linear time $\mathcal{O}(n+m)$, using the algorithms described above.

Step 2: Computing $\curvearrowleft$, $\curvearrowright$ and $\blacktriangleleft$. We sort the endpoints of the intervals given by the partial representation. Now, for every clique a, we compute $I(a)$. We search the real line from left to right and calculate $\curvearrowleft(a)$ and $\curvearrowright(a)$. At every point, we just need to remember the size of the symmetric difference of $I(a)$ and the intervals placed at the current point. This number changes only if an interval starts or ends. If the number is zero, we found a place where the clique-point a can be placed. According to it, we modify $\curvearrowleft(a)$ and $\curvearrowright(a)$.

In the end, we compute the graph of the partial ordering $\blacktriangleleft$, using $\curvearrowleft$ and $\curvearrowright$. Clearly, everything can be done in $\mathcal{O}(n^2)$.

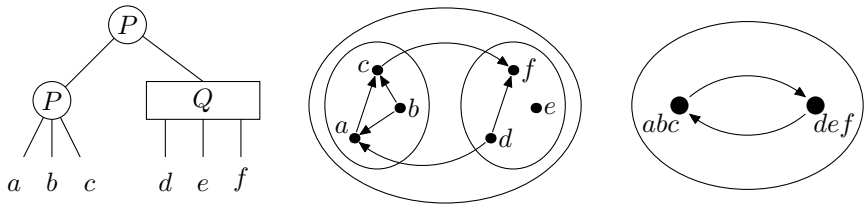


Figure 2.4: Reordering a PQ-tree, contracting cliques to big cliques.

Step 3: Finding an Ordering Extending $\blacktriangleleft$. To obtain an ordering of the cliques extending $\blacktriangleleft$, we reorder the children from the leaves to the root. When we finish the operation for a subtree, the order of the children is fixed. In the graph of $\blacktriangleleft$, for every child, we represent all the cliques contained in the subtree corresponding to this child by one big vertex—their order is never changed. When we reorder the children of a node, every child is represented by a vertex. We contract all these vertices, keeping edges in the graph, see Figure 2.4. This operation can be imagined in the following way. We cluster the given cliques, zoom out and they start to appear as one big clique, with the edges between clusters.

For a P-node, we ask whether there exists a topological ordering of the subgraph of $\blacktriangleleft$ induced by the vertices representing the children. If such an ordering exists, we use it to reorder the children. Otherwise, there exists no feasible ordering. For a Q-node, there are two possible orders. We just need to check whether one of them is permissible.

Notice the following. Every subtree has to appear consecutively in the ordering. Therefore, an ordering of a subtree does not influence orderings of the other subtrees. It means that if we can reorder a subtree compatibly to $\blacktriangleleft$, it is always a correct reordering.

This operation can be implemented in linear time depending on the size of the partial ordering $\blacktriangleleft$ which is $\mathcal{O}(n^2)$. Moreover, we can generate all the possible orderings with a polynomial delay $\mathcal{O}(n^2)$.

Step 4: Placing the Clique-points. The real line has several intervals already placed by the partial representation. We place clique-points greedily from left to right, according to the ordering. A clique-point a has to be placed in a part of the real line containing exactly all the intervals from $I(a)$. Among such points, we pick greedily the leftmost one following the last placed clique-point and place this clique-point by a small epsilon to the right. We can easily implement this greedy algorithm in time $\mathcal{O}(n)$.

Lemma 2.2. *For an ordering of the cliques compatible with the PQ-tree extending $\blacktriangleleft$, the greedy algorithm described in Step 4 never fails.*

Proof. We prove the lemma by contradiction. See Figure 2.5. Let a be a clique-point for which the procedure fails. Since a cannot be placed, there are some clique-points placed after $\curvearrowright(a)$. Let b be the leftmost one of them. If $\curvearrowleft(b) \geq \curvearrowright(a)$, we obtain $a \blacktriangleleft b$ which contradicts $b < a$. So, we know that $\curvearrowleft(b) < \curvearrowright(a)$. To show a contradiction, we question the reason the clique-point b was not placed before $\curvearrowright(a)$.

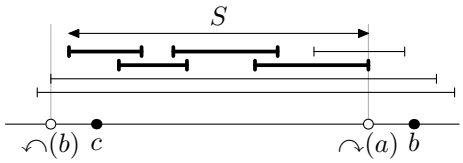


Figure 2.5: An illustration of the proof: The positions of the clique-points b and c, the intervals of S are bolder.

The clique-point b was not placed before $\curvearrowright(a)$ because all these points were blocked by some other cliques and placed intervals not in $I(b)$. Let c be one of the clique-points for which every point between c and $\curvearrowright(a)$ is already covered by a placed interval not in $I(b)$. We have a sequence S of placed intervals not contained in $I(b)$ going from c to $\curvearrowright(a)$ such that every two consecutive intervals of S intersect. This sequence S is a path in the partially represented subgraph.

Let C be a set of all the cliques containing at least one vertex from S. We show that all the cliques from C appear consecutively in the ordering of the cliques. For every interval, the cliques containing it has to appear consecutively. The consecutive intervals of S are bounded together, since they share at least one clique. Therefore, the cliques C are consecutive as well.

Now, a and c belong to C, but b does not. Since $c < b$, the consecutivity of C implies $a < b$ which is a contradiction with $b < a$. □

Step 5: Constructing a Representation. We construct a representation of the whole graph using the clique-points placed on the real line, similarly to Figure 2.1. We represent each vertex as an interval containing exactly all the clique-points corresponding to the cliques containing this vertex. Also, we slightly stretch all the intervals to ensure distinct endpoints.

The intervals placed by the partial representation contain the correct clique-points. Two vertices are contained in a common clique if and only if they are adjacent. Therefore, a pair of intervals intersects if and only if the corresponding vertices are adjacent. The obtained representation is correct.

The last step can be done in time $\mathcal{O}(n)$.

Theorem 2.3. *The algorithm solves* PREXT(**INT**) *in time* $\mathcal{O}(n^2)$. *Moreover, it can find all the different representations with a polynomial delay* $\mathcal{O}(n^2)$.

Proof. We run the described algorithm. Obviously, the conditions are necessary, otherwise the graph is not extendible. On the other hand, if they are satisfied, the algorithm constructs an ordering $\blacktriangleleft$. By Lemma 2.2, the greedy algorithm always places the clique-points on the real line and we construct a representation.

The complexity of each step was already discussed. In total, the algorithm runs in time $\mathcal{O}(n^2)$. For every other representation, we need to construct a different ordering and to run steps 4 and 5, which can be done in time $\mathcal{O}(n^2)$. □

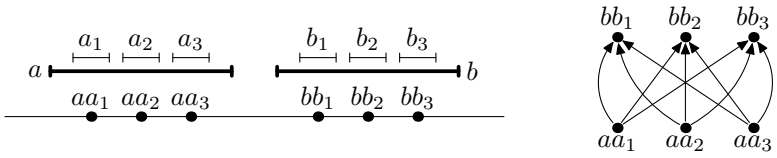


Figure 2.6: A hairbrush graph with an interval representation and the ordering $\blacktriangleleft$.

Could we do it faster? A weak point of this algorithm is that the graph created from the relation $\blacktriangleleft$ can have $\Theta(n^2)$ edges. Originally, we believed that instead of working with $\blacktriangleleft$ we could take a some smaller graph for which its transitive closure would be $\blacktriangleleft$. Notice that Step 3 would still work. Unfortunately, there exists an interval graph having $\mathcal{O}(n)$ edges with a partial representation for which we need $\Theta(n^2)$ edges to store $\blacktriangleleft$, even the transitivity does not help. In the rest of the chapter, we show this example.

A hairbrush graph is a union of two stars S_n. Denote the central vertices a and b, and the leaves $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$. The partial representation places the vertices a and b. See Figure 2.6, the represented vertices are bolder. Every maximal clique is a leaf and the corresponding central vertex. We call a clique containing a (resp. b) an a-clique (resp. an b-clique). The ordering $\blacktriangleleft$ tells that all the a-cliques are before all the b-cliques. On the other hand, we can permute a-cliques and b-cliques arbitrary. So, the graph of $\blacktriangleleft$ is $K_{n,n}$ oriented from the a-cliques to the b-cliques, having n^2 edges.

This graph is the reason why we do not believe that our algorithm can be easily optimized to run in $\mathcal{O}(n+m)$. Nevertheless, we think that it is possible to obtain a linear algorithm, probably using a different approach.

Chapter 3

Extending Proper and Unit Interval Graphs

In this section, we study two important subclasses of interval graphs. Unit interval graphs have all the intervals of a unit length. A proper interval graph has no interval contained in another interval. We denote these classes **UNIT INT** and **PROPER INT**. One can see that these restrictions are quite natural. Although the structure of these classes seems simpler than the structure of interval graphs, the first linear time recognition algorithms were discovered much later, see [LO93, HH95, Cor04].

Relation. In fact, these two classes are the same. Roberts [Rob69] proved that

$$\textbf{UNIT INT} = \textbf{PROPER INT}.$$

Clearly, every unit interval graph is also a proper interval graphs. On the other hand, we consider a representation of a proper interval graph. To get all the intervals of a unit length, we stretch and translate them without changing their topology.

Roberts proved this relation in a different way, using the following characterization: Proper/unit interval graphs are claw-free interval graphs—interval graphs without an induced subgraph $K_{1,3}$.

Extending. Surprisingly, for the problem of partial representation extension, we need to distinguish between **UNIT INT** and **PROPER INT**. There exists a partial representation, having all the intervals of a unit length, which can be extended only by proper intervals, but not by unit intervals. See Figure 3.1, the represented intervals are depicted in bold. On the other hand, every partial representation extendible by unit intervals is also extendible by proper intervals.

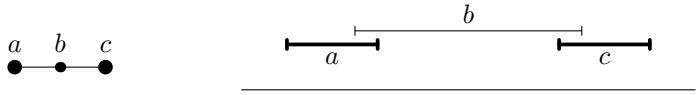


Figure 3.1: A partial representation of P_3 extensible only by proper intervals.

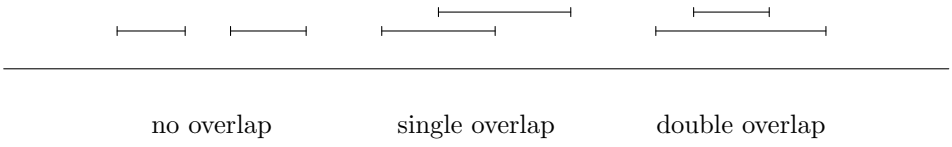


Figure 3.2: Types of interval overlapping.

First, we give an algorithm solving PREXT(**PROPER INT**) in time $\mathcal{O}(mn)$, by modifying the algorithm from Chapter 2. On the other hand, the complexity of PREXT(**UNIT INT**) remains open. We conjecture that it is also polynomially solvable. Unfortunately, we were not able to use maximal cliques to solve this problem. In the end of the chapter, we discuss unit interval graphs and relations to problems concerning Allen algebras.

Definitions. For a second, we return back to general interval graphs. For a given representation, we denote $l(x)$ (resp. $r(x)$) the left (resp. the right) endpoint of the interval x. Similarly to Chapter 2, all the intervals have distinct endpoints.

All the cliques considered in the following text are maximal. For a vertex x, we denote C_x the set of all the cliques containing x. Similarly, $C_{x,\neg y}$ denotes $C_x \setminus C_y$.

Overlapping of Intervals. A pair of intervals can overlap in three possible ways, see Figure 3.2. Formally, we say that

$$\begin{aligned} x \text{ does not overlap } y &\iff r(x) < l(y), \text{ or } r(y) < l(x), \\ x \text{ single overlaps } y &\iff l(x) < l(y) < r(x) < r(y), \text{ or } l(y) < l(x) < r(y) < r(x), \\ x \text{ double overlaps } y &\iff l(x) < l(y) < r(y) < r(x), \text{ or } l(y) < l(x) < r(x) < r(y). \end{aligned}$$

In the case of proper interval graphs, all the edges are realized by single overlaps.

Additional Conditions. Let x and y be two intervals that single overlap. Consider the ordering of the maximal cliques given by the representation. Observe that all the cliques in $C_{x,\neg y}$ appear on one side of their intersection and similarly $C_{y,\neg x}$ appears on the other side. Together with the consecutivity of C_x and C_y, we know that $C_{x,\neg y}$ and $C_{y,\neg x}$ appear consecutively in the ordering.

This introduces additional conditions to the PQ-tree. They ensure that every edge is realized by a single overlap. We show that it is everything we need to solve PREXT(**PROPER INT**). Also, notice that if both $C_{x,\neg y}$ and $C_{y,\neg x}$ are non-empty, the consecutivity conditions are already ensured.

The Modified Algorithm. Recall the algorithm from Chapter 2, we modify it to solve PREXT(**PROPER INT**). First, we check whether the given partial representation is correct. In Step 1, we introduce other restricting sets, as described above. Let xy be an edge. We add the sets $C_{x,\neg y}$ and $C_{y,\neg x}$ to appear consecutively.[1] In Step 5, we need to place the intervals in a way that all the edges are realized by single overlaps. We discuss this step now.

[1] As described above, we can skip this if the both sets are non-empty.

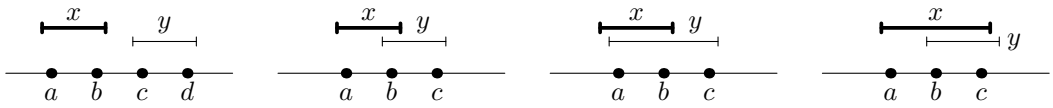


Figure 3.3: In all these cases, $x \lhd y$ for the thick interval x and the thin interval y.

Step 5: Placing Proper Intervals. Unlike interval graphs, proper interval graphs have one specific property. For every representation, the left endpoints appear in the same order as the right endpoints. We need to find such a linear ordering $\lhd$.

Conditions for $\lhd$ are given by the ordering of the cliques and by the partial representation:

Conditions for Different Cliques. If two vertices are contained in different cliques, their order in $\lhd$ is fixed, see Figure 3.3. We denote $\leftarrow(x)$ (resp. $\rightarrow(x)$) the leftmost (resp. the rightmost) clique containing the vertex x. We order the vertices lexicographically, according to $\leftarrow$ and $\rightarrow$:

$$\forall x, y \qquad \leftarrow(x) < \leftarrow(y) \vee \big(\leftarrow(x) = \leftarrow(y) \wedge \rightarrow(x) < \rightarrow(y)\big) \implies x \lhd y.$$

This orders every pair of vertices x and y having $C_x \neq C_y$. Since additional conditions were introduced to the PQ-tree, no two vertices x and y satisfy

$$\leftarrow(x) < \leftarrow(y) < \rightarrow(y) < \rightarrow(x).$$

Conditions for the Same Cliques. If two vertices x and y represented by the partial representation are contained in the same cliques, we order them according to the partial representation. Notice that if they would be contained in different cliques, their order would be correctly given by the conditions described above.

Clearly, there exists a linear ordering $\lhd$ satisfying all the conditions. To construct a representation, we use $\lhd$ in the following way. We order all the endpoints around corresponding clique-points, see Figure 3.4. Some of the endpoints are already fixed. We place the other left endpoints to the left of the corresponding clique-points, in the order given by $\lhd$. Similarly, we place the other right endpoints to the right of the corresponding clique-points according to $\lhd$.

Lemma 3.1. *The representation constructed in Step 5 is a correct proper interval representation of the graph.*

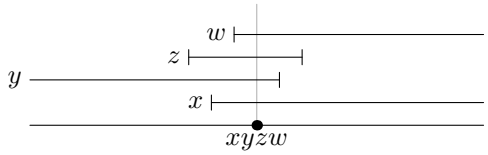


Figure 3.4: Placing proper intervals around a clique-point, according to the ordering $y \lhd z \lhd x \lhd w$.

Proof. A pair of intervals intersects if and only if there exists a common clique containing the both vertices. We need to discuss that all the intersections are realized by single overlaps.

Let x and y be two vertices. If they are contained in different cliques, we obtain a condition to $\lhd$ according to the cliques. In such a case, we always place intervals correctly. If they are contained in the same clique, their left endpoints are placed in the same order as their right endpoints. □

We claim that if the graph is extendible, the algorithm does not fail and produces a correct representation.

Theorem 3.2. *The algorithm will solve* PREXT(**PROPER INT**)*, the running time is* $\mathcal{O}(mn)$*. We can find all the different representations with a polynomial delay* $\mathcal{O}(n^2)$*.*

Proof. Let the partial representation be extendible. There exists an ordering of the cliques satisfying the additional sets introduced in Step 1. The original algorithm from Chapter 2 does not fail and places the cliques on the real line. By Lemma 3.1, we construct a correct representation in Step 5.

On the other hand, if the partial representation is not extendible, the algorithm fails somewhere before Step 5.

The time $\mathcal{O}(mn)$ is necessary for the additional sets introduced to the PQ-tree. We introduce m additional sets, each having at most n elements. The most time-consuming part of the algorithm is the construction of a PQ-tree. When we search for more solutions, we construct the PQ-tree only ones. After that we generate different ordering of the maximal cliques. For every ordering, we construct $\lhd$ and a representation in time $\mathcal{O}(n^2)$. □

Unit Interval Graphs. Since the result of Roberts [Rob69], not much work was done in the direction of unit interval graphs. Mostly, the equivalent class of proper interval graphs was considered since it is much easier to work with. Also, there are several other proofs that **UNIT INT** = **PROPER INT**, for example [BW99, LSS09]. In all the papers, the intervals are stretched and shifted to obtain a unit interval representation. This approach does not appear to be useful in solving PREXT(**UNIT INT**), since the fixed intervals cannot be transformed.

When we try to extend a unit interval graph, we have to consider the spaces between the placed intervals. As Figure 3.1 shows, the extension can be restricted by too small or too big spaces.

One could expect that an optimization program could find a feasible solution. But since we do not know the order of intervals from left to right, we were not able to construct a linear program. We obtain the following second-order cone program.

Let $v_1, \ldots, v_k$ be the vertices placed by the partial representation and $v_{k+1}, \ldots, v_n$ be the rest of the vertices. To every vertex v_i, we assign a variable x_i representing

the position of its left endpoint in the constructed representation. We want to find a solution to:

$$\begin{aligned} x_i &= l(v_i) && \forall i \in \{1,\dots,k\}, \\ x_i &\in \mathbb{R} && \forall i \in \{k+1,\dots,n\}, \\ |x_i - x_j| &\leq 1 && \forall i,j : v_iv_j \in E(G), \\ |x_i - x_j| &> 1 && \forall i,j : v_iv_j \notin E(G) \end{aligned}$$

where the last two conditions can be rewritten to a quadratic form:

$$\begin{aligned} (x_i - x_j)^2 = x_i^2 - 2x_ix_j + x_j^2 &\leq 1 && \forall i,j : v_iv_j \in E(G), \\ (x_i - x_j)^2 = x_i^2 - 2x_ix_j + x_j^2 &> 1 && \forall i,j : v_iv_j \notin E(G). \end{aligned}$$

We do not know whether this program can be solved in polynomial time.

Originally, we have believed that the following procedure would work. We start by solving the problem for proper interval graphs. In Step 5, we use the ordering $\lhd$ to remove the absolute values. For every pair of vertices, we obtain a linear inequality since we know which interval is the left one. This linear program can be solved in polynomial time. Unfortunately, we do not know how to find a correct ordering. Some orderings of the maximal cliques can be not extendible, but it does not imply that the graph is not extendible. We were not able to find a restriction which would allow us to find a correct ordering $\lhd$ in polynomial time.

We strongly believe that PREXT(**UNIT INT**) is solvable in polynomial time, but its complexity remains open.

Allen Algebras. As a generalization of PREXT(**PROPER INT**), we can specify for every edge which overlapping relation should realize it. Or we could generalize even further and obtain problems introduced by Allen [All83]. Surprisingly, we independently obtained a very similar notion as Allen which only shows how natural these problems are.

These problems are studied in theory of artificial intelligence and time reasoning. Golumbic [Gol98] wrote a survey a survey about time reasoning. In the rest of the chapter, we sketch these problems.

We have several events. Every event can be represented by an interval in the timeline. Allowing shared endpoints, Allen characterized thirteen *primitive relations* between the events, see Figure 3.5. A *relation* is a union of several primitive relations. For some pairs of events, we specify relations in which they can occur. For example, we can specify for events x and y that either x is before y, or x is during y. We want to find intervals representing the events such that all the relations are satisfied. This is called the *interval satisfiability problem* (ISAT).

Vilian and Kautz [VK86] proved that ISAT is NP-complete. Golumbic and Shamir [GS93] gave a more simple proof, using the interval graph sandwich problem.

We can restrict the problem that only some relations are allowed to be used—it gives 2^{8192} $(= 2^{2^{13}})$ different problems. Notice that allowing additional relations

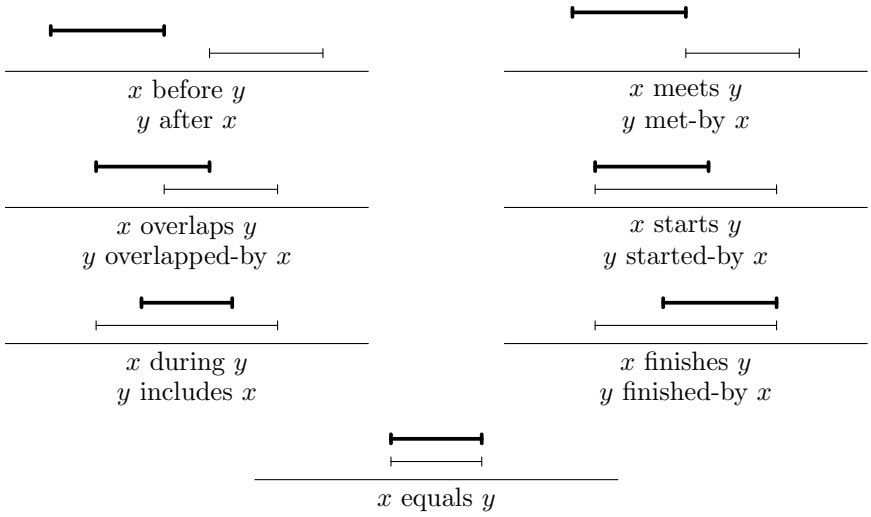


Figure 3.5: Thirteen primitive relations between the thick interval x and the thin interval y.

makes the problem only harder. On the other hand, if a problem is proved to be polynomially solvable, the more restricted problems are also polynomially solvable. We obtain an inclusion ordering of all these problems. After twenty years of intense studies, a dichotomy was proved. By results [NB95, DJ96], all eighteen maximal subsets of relations for which the problem is polynomially solvable were identified. Later, Krokhin, Jeavons and Jonsson [KJJ03] proved that the ISAT problem is NP-complete for all the subsets above them.

We can describe the problems PREXT(**INT**) and PREXT(**PROPER INT**) using these conditions. In the case of PREXT(**INT**), we do it in the following way. If vertices are non-adjacent, we assign them the relation $\{\text{before}, \text{after}\}$. If they are adjacent, we assign the complement of the previous relation. For intervals fixed by a partial representation, we give singleton relations. Unfortunately, since we specify the relations for only some pairs of the vertices, the more general ISAT problem obtained by allowing these relations is NP-complete. For an unspecified pair, we do not even know whether the corresponding intervals intersect.

Golumbic [GS93] considered a situation when a relation is given for every pair of events. In this setting, more problems are solvable in polynomial time. For example, RECOG(**INT**) can be described in this way. We are not familiar with any result in this direction that would give a polynomial solution for PREXT(**INT**) or PREXT(**PROPER INT**).

Chapter 4

Extending Comparability, Permutation and Function Graphs

In this chapter, we consider the partial representatiton extension problem for three classes of graphs with surprising connections.

For a graph G, let $\overline{G}$ denote complement of G. For a class $\mathcal{C}$, we define the complementary class $\mathbf{co}\,\mathcal{C}$:

$$\mathbf{co}\,\mathcal{C} = \{\overline{G} \mid G \in \mathcal{C}\}.$$

Comparability Graphs. For an orientation of a graph, we denote an edge xy oriented from x to y by $x \to y$. We call an orientation transitive, if for every two edges $x \to y$ and $y \to z$ there exists an edge $x \to z$. Comparability graphs are undirected graphs which admit transitive orientations. In other words, every comparability graph is a graph of a partial order for which we forget the orientation of the edges. We denote the class of comparability graphs by **CO**.

Recognition of CO. Currently, the fastest recognition algorithm [MS99] constructs a transitive orientation in time $\mathcal{O}(n+m)$ if the graph is a comparability graph. But we need to verify whether the orientation is transitive which can be done using matrix multiplication, currently in time $\mathcal{O}(n^{2.376})$. We describe a simpler algorithm in time $\mathcal{O}(\Delta \cdot m)$, due to Golumbic [Gol77].

We define the relation Γ on the edges. Let x, y and z be vertices. We write $xy\ \Gamma\ yz$ if and only if $xy \in E$, $yz \in E$ but $xz \notin E$. The algorithm is based on the following key observation:

Observation 4.1. *Let x, y and z be vertices of a comparability graph, such that $xy\ \Gamma\ yz$. Then every transitive orientation of the graph orients the both xy and yz either to y, or from y.*

The algorithm work in this way. We start with an undirected graph. We repeat the following step till all the edges are oriented. In the beginning, the algorithm picks an arbitrary non-oriented edge and orients it in an arbitrary direction. This maybe forces several other edges to be oriented in some directions. According to transitivity

and the relation Γ, we orient every edge for which the direction is now given. If we are forced to change the direction of an edge, the algorithm fails.

Golumbic proved that this algorithm fails if and only if the graph is not a comparability graph, independently on the choices of the edges in the first part of every step. A straight-forward implementation of this algorithm works in time $\mathcal{O}(\Delta \cdot m)$. We need to orient every edge, which makes $\mathcal{O}(m)$ steps. After orienting any edge we check all the incident edges whether some other orientations are forced, which can be done in $\mathcal{O}(\Delta)$. Also, this algorithm allows to find other transitive orientations with a polynomial delay $\mathcal{O}(\Delta \cdot m)$, by choosing different orientations of edges in the beginning of every step.

Extending CO. First, we describe what we mean by a representation of a comparability graph. A representation is a transitive orientation of the edges. A partial representation is an orientation of some edges. Therefore, the problem PREXT(**CO**) asks whether it is possible to orient the rest of the edges to obtain a transitive orientation. We solve this problem by modifying the algorithm described above. A similar approach was used by Golumbic [GS93], as a part of the algorithm solving one problem concerning Allen algebras which were described in Chapter 3.

Let E' be the set of the edges oriented by a partial representation. In the beginning of every step, we can choose an arbitrary edge and orient it in an arbitrary direction. The only change is that we pick edges from E' first. We start with an undirected graph. If there is still an edge from E' which is not oriented, we choose this edge and orient it in the direction given by the partial representation. Then we proceed the rest of the step unchanged. If we are forced to orient an edge from E' in the opposite direction, the algorithm fails. The algorithm gives a transitive orientation of the graph, extending the orientation given by the partial representation.

Theorem 4.2. *The described algorithm solves* PREXT(**CO**) *in time* $\mathcal{O}(\Delta \cdot m)$. *Moreover, it can find all the transitive orientations with a polynomial delay* $\mathcal{O}(\Delta \cdot m)$.

Proof. We only specify which edges are chosen in the beginning of every step. Since the algorithm of Golumbic is correct, this algorithm is also correct. If the algorithm fails trying to reorient an edge, the graph is not a comparability graph. If we are forced to orient an edge from E' in a bad direction, this was forced by some other edges in E' and the partial representation cannot be extended.

We can implement the algorithm in the similar manner to the above one, in time $\mathcal{O}(\Delta \cdot m)$. The algorithm allows to find all the different transitive orientations. The direction of E' are forced, the other edges can be oriented arbitrary. □

Permutation Graphs. A permutation graph is given by a permutation π of the elements $\{1, \ldots, n\}$. The vertices of the graph are the elements of the permutation. Two vertices x and y, $x < y$, are adjacent if and only if $\pi(x) > \pi(y)$. A pair of adjacent vertices is called an *inversion* of the permutation. We denote the class of permutation graphs by **PERM**.

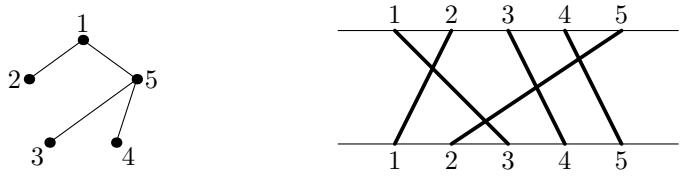


Figure 4.1: A permutation graph of the permutation $\pi = (3, 1, 4, 5, 2)$ with an intersection representation by segments touching two parallel lines.

We can represent a permutation graph as an intersection graph of segments in the plane, see Figure 4.1. We place two copies of points $\{1, \ldots, n\}$ on two parallel lines. To a vertex x, we assign a segment from x on the top line to $\pi(x)$ on the bottom line. It is easy to show that permutations graphs are exactly intersection graphs of segments in the plane with distinct ends touching two parallel lines.

Recognition of PERM. Their recognition is using comparability graphs, based on the following characterization, by Even, Pnueli and Lempel [EPL72]:

$$\mathbf{PERM} = \mathbf{CO} \cap \mathbf{coCO}.$$

To recognize a permutation graph, it is sufficient to check whether both this graph and its complement are comparability graphs. First, by constructing a segment representation, we show that every graph from $\mathbf{CO} \cap \mathbf{coCO}$ is also a permutation graph.

Let G be a graph from $\mathbf{CO} \cap \mathbf{coCO}$. We can transitively orient its edges $\overrightarrow{E_1}$ and the edges of its complement $\overrightarrow{E_2}$. Together, the orientations $\overrightarrow{E_1}$ and $\overrightarrow{E_2}$ form a transitive orientation of a complete graph—a linear ordering. See Figure 4.2.

Now, we have two parallel lines in the plane between which we want to place segments representing the vertices of the graph. On the top line, we place points $A_1, \ldots, A_n$ representing the vertices in the linear ordering given by $\overrightarrow{E_1} \cup \overrightarrow{E_2}$. For the bottom line, we reverse the orientation $\overrightarrow{E_1}$ and place points $B_1, \ldots, B_n$ according to the linear ordering $\overleftarrow{E_1} \cup \overrightarrow{E_2}$. To a vertex v, we assign the segment $A_v B_v$.

We need to verify that these segments give an intersection representation of the graph. If two vertices are adjacent, their orders on the top line and on the bottom are different and the corresponding segments intersect. On the other hand, if two vertices are non-adjacent, their orders are the same and the segments do not intersect. So, we obtain a valid intersection representation of the graph G which implies G is a permutation graph.

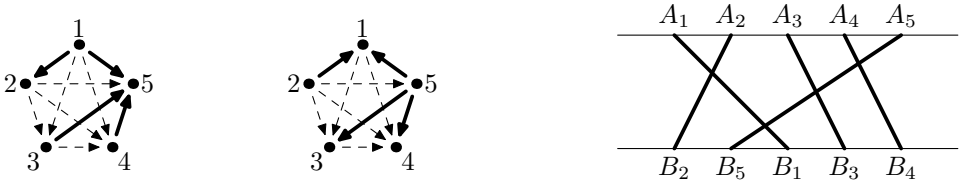


Figure 4.2: Constructing a segment representation from orientations $\overrightarrow{E_1}$ and $\overrightarrow{E_2}$.

To obtain the other inclusion, notice that a segment representation gives transitive orientations $\overrightarrow{E_1}$ and $\overrightarrow{E_2}$. Another way is to observe that **PERM** = **coPERM** (by reversing the bottom line) and **PERM** $\subseteq$ **CO** (similarly to function graphs described below).

Extending PERM. A partial representation places several segments between two parallel lines. The problem PREXT(**PERM**) asks whether it is possible to add the rest of the segments to obtain a representation of a given permutation graph. Notice that all the segments have distinct endpoints.

Let G be an input graph. This graph is a permutation graph if and only if G and $\overline{G}$ are comparability graphs. The partial representation fixes directions of several edges of G and $\overline{G}$. We orient edges according to the ordering of the endpoints on the top line from left to right. If two segments intersect, the corresponding edge is oriented in G, otherwise in $\overline{G}$.

For the partially oriented G and $\overline{G}$, we run the algorithm for comparability graph extension which is described above. If extending is not possible, the algorithm fails. Otherwise, we obtain transitive orderings $\overrightarrow{E_1}$ and $\overrightarrow{E_2}$. We place the rest of the points $A_1, \ldots, A_n$ and $B_1, \ldots, B_n$ in the correct order, and construct a representation of G as described above—a vertex v is represented by a segment $A_v B_v$.

Theorem 4.3. *The described algorithm solves* PREXT(**PERM**) *correctly, the running time is* $\mathcal{O}(\Delta \cdot m)$. *We can find all the representations with a polynomial delay* $\mathcal{O}(\Delta \cdot m)$.

Proof. If G has a segment representation extending the partial representation, then this representation does not change the order of the placed segments. Therefore, the corresponding transitive orientations $\overrightarrow{E_1}$ and $\overrightarrow{E_2}$ have the edges oriented according to the partial representation. It is possible to extend the partially oriented G and $\overline{G}$. Using these orientations, we construct a representation.

The running time is $\mathcal{O}(\Delta \cdot m)$ since we need to run two instances of the algorithm solving PREXT(**CO**). To find all the representations, we use that the algorithm for PREXT(**CO**) has the same property. □

Function Graphs. Function graphs are a natural generalization of permutation graphs. They are intersection graphs of continuous functions $f : [0, 1] \to \mathbb{R}$. In other words, they are intersection graphs of curves touching two vertical lines which have exactly one intersection with every vertical line between them. We denote function graphs by **FUN**.

Recognition of FUN. Similarly to permutation graphs, function graphs and comparability graphs are connected, due to Golumbic, Rotem and Urrutia [GRU83]:

$$\textbf{FUN} = \textbf{coCO}.$$

So we can recognize them easily in time $\mathcal{O}(\Delta \cdot m)$, using the recognition algorithm for comparability graphs.

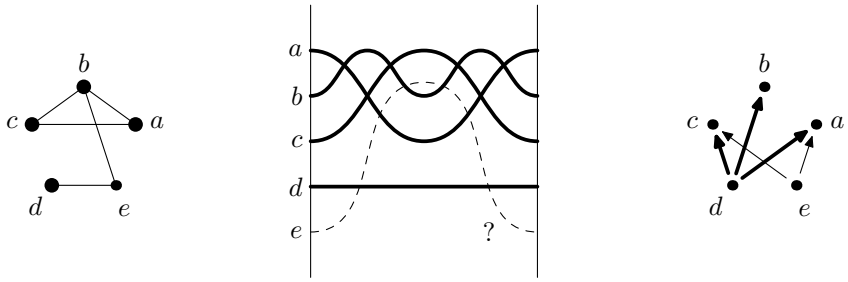


Figure 4.3: A function graph G with a partial representation which is not extendible, but the complement $\overline{G}$ with the corresponding partial orientation which is extendible. The functions/oriented edges given by the partial representations are bolder.

We sketch why **FUN** $\subseteq$ **coCO**. There is a natural ordering between non-adjacent vertices. Non-adjacent vertices are represented by two non-intersecting functions, so we orient the edge in the complement from the bottom function to the top function. Clearly, we obtain a partial ordering.

Extending FUN. The complexity of PREXT(**FUN**) remains open. In the rest of the chapter, we show why we cannot apply an approach similar to PREXT(**PERM**).

In Figure 4.3, there is a function graph G with a partial representation which is not extensible. On the other hand, if we construct the corresponding partial orientation of the complement $\overline{G}$, we can construct a transitive orientation extending this partial orientation. So, $\overline{G}$ is a comparability graph.

Notice that the edges of $\overline{G}$ do not represent every information given by the partial representation of G. It is true that neither a, nor c separate b from e. But a and c together separate b from e. We could capture this by a hyperedge from a and c to b. In such a case, the transitivity of a orientation would require an edge from e to b. But this edge is missing and the partial representation cannot be extended. It is not clear whether it is possible to find all these hyperedges in polynomial time and whether the work of Golumbic could be extended to hypergraphs.

Conclusions

We conclude the thesis with related questions and several open problems.

We considered the problem of partial representation extension for several classes of graphs. We modified several recognition algorithms to solve these problems. We were able extend interval graphs (Chapter 2), proper interval graphs (Chapter 3), comparability graphs and permutation graphs (Chapter 4).

On the other hand, several problems remain open:

Question 1: Extending UNIT INT. What is the complexity of PREXT(**UNIT INT**)? More details about this problem are written in Chapter 3.

Question 2: Extending FUN. What is the complexity of PREXT(**FUN**)? As a more simple case, we could consider segment representation (similar to permutation graphs) where segments touching two parallel lines can share endpoints. More details about the problem are written in Chapter 4.

Question 3: Extending string graphs. Recognition of string graphs (**STRING**) is NP-complete, but it is non-trivial to prove that it belongs to NP. Extending of string graphs is clearly NP-hard. Is this problem also NP-complete? It would be interesting to find out whether it is possible to generalize the proof [SSŠ02] that RECOG(**STRING**) belongs to NP.

Question 4: Extending other classes. There are several other classes of intersection graphs which can be recognized in polynomial time. For example, circular arc graphs can be recognized in time $\mathcal{O}(n + m)$ [McC01] and circle graphs can be recognized in time $\mathcal{O}(n^2)$ [Spi94]. What is the complexity of their extending? More information about these classes are contained in books [Gol04, Spi03].

Question 5: Faster algorithm. Can we solve the partial representation extension problem for the classes described in this thesis faster? There is no reason to believe that algorithms described here are optimal. For example, for most of the problems solvable for interval graphs in polynomial time, there are known linear time algorithms. We believe that it should be possible to construct faster algorithms, maybe using different techniques.

Bibliography

[AB09] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[ABF+10] P. Angelini, G. D. Battista, F. Frati, V. Jelínek, J. Kratochvíl, M. Patrignani, and I. Rutter. Testing Planarity of Partially Embedded Graphs. In *SODA '10: Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, 2010.

[All83] J. F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983.

[Ben59] S. Benzer. On the topology of the genetic fine structure. *Proc. Nat. Acad. Sci. U.S.A.*, 45:1607–1620, 1959.

[BL76] K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and planarity using PQ-tree algorithms. *Journal of Computational Systems Science*, 13:335–379, 1976.

[BM04] J. M. Boyer and W. J. Myrvold. On the cutting edge: Simplified $\mathcal{O}(n)$ planarity by edge addition. *Journal of Graph Algorithms and Applications*, 8:241–273, 2004.

[BW99] K. P. Bogart and D. B. West. A short proof that "proper = unit". *Discrete Math.*, 201(1-3):21–23, 1999.

[Cor04] D. G. Corneil. A simple 3-sweep LBFS algorithm for the recognition of unit interval graphs. *Discrete Appl. Math.*, 138(3):371–379, 2004.

[COS09] D. G. Corneil, S. Olariu, and L. Stewart. The LBFS Structure and Recognition of Interval Graphs. *SIAM Journal on Discrete Mathematics*, 23(4):1905–1953, 2009.

[DJ96] T. Drakengren and P. Jonsson. Eight Maximal Tractable Subclasses of Allen's Algebra with Metric Time. *Journal of Artificial Intelligence Research*, 7:25–45, 1996.

[EPL72] S. Even, A. Pnueli, and A. Lempel. Permutation Graphs and Transitive Graphs. *J. ACM*, 19(3):400–410, 1972.

[FG65] D. R. Fulkerson and O. A. Gross. Incidence matrices and interval graphs. *Pac. J. Math.*, 15:835–855, 1965.

[GI95] D. S. Greenberg and S. Istrail. Physical mapping by STS hybridization: algorithmic strategies and the challenge of software evaluation. *J. Comput. Biol.*, 2(2):219–274, 1995.

[Gol77] M. C. Golumbic. The complexity of comparability graph recognition and coloring. *J. Combin. Theory, Ser. B*, 22:68–90, 1977.

[Gol98] M. C. Golumbic. Reasoning about time. In *Mathematical Aspects of Artificial Intelligence, F. Hoffman, ed.*, volume 55, pages 19–53, 1998.

[Gol04] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. North-Holland Publishing Co., 2004.

[GRU83] M. C. Golumbic, D. Rotem, and J. Urrutia. Comparability graphs and intersection graphs. *Discrete Mathematics*, 43(1):37–46, 1983.

[GS93] M. C. Golumbic and R. Shamir. Complexity and algorithms for reasoning about time: a graph-theoretic approach. *J. ACM*, 40(5):1108–1133, 1993.

[Haj57] G. Hajós. Über eine art von graphen. *Internationale Mathematische Nachrichten*, 11:65, 1957.

[HH95] P. Hell and J. Huang. Lexicographic orientation and representation algorithms for comparability graphs, proper circular arc graphs, and proper interval graphs. *Journal of Graph Theory*, 20(3):361–374, 1995.

[HT74] J. Hopcroft and R. Tarjan. Efficient Planarity Testing. *J. ACM*, 21(4):549–568, 1974.

[KJJ03] A. Krokhin, P. Jeavons, and P. Jonsson. Reasoning about temporal relations: The tractable subalgebras of Allen's interval algebra. *J. ACM*, 50(5):591–640, 2003.

[Kra91] Jan Kratochvíl. String graphs. II. recognizing string graphs is NP-hard. *Journal of Combinatorial Theory, Series B*, 52(1):67–78, 1991.

[Kur30] K. Kuratowski. Sur le problème des courbes gauches en topologie. *Fund. Math.*, 15:217–283, 1930.

[LO93] P. J. Looges and S. Olariu. Optimal greedy algorithms for indi erence graphs. *Comput. Math. Appl.*, 25:15–25, 1993.

[LSS09] M. C. Lin, F. J. Soulignac, and J. L Szwarcfiter. Short Models for Unit Interval Graphs. *Electronic Notes in Discrete Mathematics*, 35:247–255, 2009.

[Mar45] E. S. Marczewski. Sur deux propriétés des classes d'ensembles. *Fund. Math.*, 33:303–307, 1945.

[McC01] R. McConnell. Linear-Time Recognition of Circular-Arc Graphs. In *FOCS '01: Proceedings of the 42nd IEEE symposium on Foundations of Computer Science*, page 386, 2001.

[MM99] T. A. McKee and F. R. McMorris. *Topics in Intersection Graph Theory*. SIAM Monographs on Discrete Mathematics and Applications, 1999.

[MPT98] J. Meidanis, O. Porto, and G. P. Telles. On the consecutive ones property. *Discrete Applied Mathematics*, 88(1-3):325–354, 1998.

[MS99] R. M. McConnell and J. P. Spinrad. Modular decomposition and transitive orientation. *Discrete Math.*, 201(1-3):189–241, 1999.

[NB95] B. Nebel and H.-J. Bürckert. Reasoning about temporal relations: a maximal tractable subclass of Allen's interval algebra. *J. ACM*, 42(1):43–66, 1995.

[Pap94] C. M. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[Pat06] M. Patrignani. On Extending a Partial Straight-Line Drawing. In *Lecture Notes in Computer Science*, volume 3843, pages 380–385, 2006.

[Rob69] F. S. Roberts. Indifference graphs. In *F. Harary (Ed.), Proof Techniques in Graph Theory*, pages 139–146. Academic Press, 1969.

[Rob76] F. S. Roberts. *Discrete Mathematical Models, with Applications to Social, Biological, and Environmental Problems*. Prentice-Hall, Englewood Cliffs, 1976.

[RTL76] D. J. Rose, R. E. Tarjan, and G. S. Lueker. Algorithmic Aspects of Vertex Elimination on Graphs. *SIAM Journal on Computing*, 5(2):266–283, 1976.

[Sch90] W. Schnyder. Embedding planar graphs on the grid. In *SODA '90: Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 138–148, 1990.

[Spi94] J. Spinrad. Recognition of Circle Graphs. *Journal of Algorithms*, 16(2):264–282, 1994.

[Spi03] J. P. Spinrad. *Efficient Graph Representations*. Field Institute Monographs, 2003.

[SSŠ02] M. Schaefer, E. Sedgwick, and D. Štefankovič. Recognizing string graphs in NP. In *STOC '02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 1–6, 2002.

[Sto68] K. E. Stoffers. Scheduling of traffic lights–a new approach. *Transportation Research*, 2:199–234, 1968.

[VK86] M. Vilian and H. Kautz. Constraint propagation algorithms for temporal reasoning. In *Proc. Fifth Nat'l. Conf. on Artificial Intelligence*, pages 337–382, 1986.