The Complexity of Greedy Algorithms on Ordered Graphs

Thesis submitted for the degree of
Doctor of Philosophy
at the University of Leicester

Antonio Puricella

Department of Mathematics
and Computer Science
University of Leicester

August 2002

The Complexity of Greedy Algorithms on Ordered Graphs

Antonio Puricella

Abstract

Let π be any fixed polynomial time testable, non-trivial, hereditary property of graphs. Suppose that the vertices of a graph G are not necessarily linearly ordered but partially ordered, where we think of this partial order as a collection of (possibly exponentially many) linear orders in the natural way. In the first part of this thesis, we prove that the problem of deciding whether a lexicographically first maximal (with respect to one of these linear orders) subgraph of G satisfying π, contains a specified vertex is **NP**-complete. For some of these properties π we then show that by applying certain restrictions the problem still remains **NP**-complete, and show how the problem can be solved in deterministic polynomial time if the restrictions imposed become more severe.

Let H be a fixed undirected graph. An H-colouring of an undirected graph G is a homomorphism from G to H. In the second part of the thesis, we show that, if the vertices of G are partially ordered then the complexity of deciding whether a given vertex of G is in a lexicographically first maximal H-colourable subgraph of G is **NP**-complete, if H is bipartite, and $\mathbf{\Sigma_2^p}$-complete, if H is non-bipartite. We then show that if the vertices of G are linearly, as opposed to partially, ordered then the complexity of deciding whether a given vertex of G is in the lexicographically first maximal H-colourable subgraph of G is **P**-complete, if H is bipartite, and $\mathbf{\Delta_2^p}$-complete, if H is non-bipartite.

In the final part of the thesis we show that the results obtained can be parallelled in the setting of graphs where orders are given by degrees of the vertices.

Acknowledgements

I would like to thank my supervisor, Professor Iain Stewart, for all the support, guidance and advice provided during the preparation of this thesis. I would also like to thank my family, for being a constant source of encouragement and for always being there for me when I needed them.

Contents

List of Figures

Chapter 1

Introduction

1.1 A bit of history

The development of modern computers started in the 1930s, and their speed has consistently increased to achieve levels of performance that would have been unimaginable only decades ago. The history of this development is fascinating, and the interested reader can find more about it in many books, for example [8, 41]. Of course even the most powerful computer would be useless without programs and the underlying algorithms. Although modern computers really came into existence only decades ago, the idea of an algorithm is much older.

The word *algorithm* derives from the name of the mathematician Mohammed al-Khowârizmî, who lived in the ninth century and devised precise rules for the addition, subtraction, multiplication and division of decimal numbers. The name was translated into Latin as Algorismus, and then it became, in English, algorithm [22]. An algorithm is normally defined as a set of rules for calculation, to be carried out either by hand, or more often, on a machine [7].

In this thesis we will be interested in algorithms related to graph theory, an area of discrete mathematics that has provided researchers with interesting problems for several centuries. One of the first graph-theoretic problems is known as the Königsberg Bridge problem. There were two islands in Königsberg, linked to each other and to the banks of the Pregel River by seven bridges. The problem was to cross each bridge exactly once starting from any of the four land areas, and to return to the start point [21]. The Königsberg Bridge problem was solved by Leonhard Euler [12], who proved, by modelling the problem using a mathematical structure called a graph, that this is not possible.

A graph is basically a diagram consisting of points, called vertices, joined by lines, called edges, and such that each edge joins exactly two vertices (a precise definition will be given in the following chapter). Virtually every problem that can be represented as a collection of objects that are somehow related to each other can be modelled by a graph, by assigning a vertex to each element and joining two vertices with an edge if the two corresponding elements are related. Graphs have been used to model all sorts of real life situations, from computer networks, roadways and databases to the world wide web, and research in the area of graph theory is in constant development.

1.2 Efficient algorithms

It is common practice to consider algorithms that terminate their execution in a time polynomial in the size of the input instance as efficient, and algorithms that terminate in a time which is not a polynomial, say exponential or worse, as inefficient (of course, an algorithm need not even terminate). One point should be made clear; that is, that the complexity of a problem relates to its worst case instance; that is, some (possibly pathological) instance for which the known algorithm requires the most time. The performance of the algorithm might be much better on average, but in this thesis we will only consider the worst case scenario.

All problems for which efficient algorithms are known belong to the complexity class **P**, while many problems for which the only known algorithms require exponential time belong to **NP**. Both classes **P** and **NP** contain problems that seem to represent the complexity of the class: these problems, known as **P**-complete and **NP**-complete problems respectively, are such that every other problem in the class can be translated into them, and if these complete problems can be solved more efficiently then so can all problems in the respective class.

With the constant increase in computational power, it might seem reasonable to assume that all problems can now be solved by a computer, and that there is no real need to worry about the efficiency of algorithms anymore. As it is explained in almost every book on the theory of algorithms, see for example [7, 22], this is not the case. Inefficient algorithms can solve only instances of limited size even on the most powerful machines, and many problems in $\mathbf{NP}$ are therefore (under the assumption that $\mathbf{P} \neq \mathbf{NP}$) considered intractable.

There are thousands of problems for which efficient algorithms are not known, and the list is ever growing. Indeed books have been written in this regard [15]. These problems are not only interesting from a theoretical point of view, but relate to practical problems, and they occur in areas like cryptography, DNA sequence analysis, operations research, facility location problems and so on. A common example, often used because it is as easy to understand as it is difficult to solve, is that of the Travelling Salesman problem. A salesman has to visit a number of cities, and return to the start point, in such a way that each city is visited exactly once, and the distance travelled is as small as possible. Given the distance between any two cities, what route should he choose? Such a problem can be modelled by a weighted graph, where cities are represented by the vertices and there is an edge between any two vertices. Each edge has an associated weight, which represents the distance between the two corresponding cities. This is an example of an *optimisation* problem; that is, a problem where the objective is to find the best of all possible solutions. No polynomial time algorithm to solve the Travelling Salesman problem is known.

Even if we do not know how to do so efficiently, it is still often necessary to solve this kind of problem in practice. To quickly obtain a solution on any instance, some relaxations can be made. For example, it might be possible to devise a simple algorithm that efficiently produces a good or possibly even optimal solution to the problem in certain cases, but which returns a solution that is far from optimal in others. This kind of algorithm is generally known as a *heuristic algorithm*: a simple set of rules is applied to quickly solve the problem but where there is no guarantee on the quality of the solution returned.

Heuristic algorithms have been devised for numerous problems. Going back to the example of the Travelling Salesman, the following algorithm can be used. Start at a given city, and then, at each step, visit the closest unvisited city. When all cities have been visited, return to the starting point. The performance of this algorithm is not far from optimal on some instances, and the algorithm quickly returns a solution on every instance, but there are cases for which the route chosen is much longer than the shortest possible one (for more details see [7]).

The kind of heuristic algorithms that we will be interested in are known as *greedy algorithms*, and they work by making, at any stage, an 'obvious' choice (generally based upon maximising or minimising some parameter) and selecting the element that appears more promising at the moment. In some sense, they can be considered as local algorithms. They are generally quick to devise, always return an answer, and they are efficient, but are not always guaranteed to return the optimal answer. We give an example of this method by considering how it is possible to obtain a solution to a very well known graph theoretical problem, again not known to be solvable in polynomial time, called the maximum independent set problem. An independent set is a collection of vertices of a graph that are pairwise nonadjacent. The problem consists of finding the independent set of largest size in a given graph. By relaxing our expectations it is possible to devise a greedy algorithm that finds a maximal independent set of vertices, that is, a collection of vertices from the graph, all pairwise nonadjacent, such that every other vertex in the graph, which is not in the set, is adjacent to at least one vertex in the set. Such an independent set is called maximal, as opposed to maximum, because it is not necessarily the largest possible one. The algorithm takes as input an undirected graph with the vertices labelled 1, 2, $\ldots, n$, where n is the number of vertices in the graph. It greedily computes a maximal independent set of vertices. The algorithm is as follows.

Greedy Maximal Independent Set algorithm

begin

$S := \emptyset$

for $i := 1$ **to** n

if vertex i is not adjacent to any vertex in S **then** $S := S \cup \{i\}$

end

The algorithm examines the vertices according to the linear order given by the labels on the vertices. At each stage it tries to add to the set of chosen vertices the lowest numbered untried vertex. This algorithm always returns an answer in polynomial time, but the size of the independent set depends on the order, that is, the labels, given to the vertices. For at least one order the solution returned will be the maximum independent set, while for others its size could be far from optimal (for more details see [19]).

Even when efficient algorithms for a problem exist, and are practically useful, given the availability of multi-processor computers, it is only natural to wonder whether such algorithms can be transformed into much faster ones that take full advantage of the number of processors at their disposal. The kind of speedup that is sought, in this case, should bring the running time of the algorithm from a polynomial in the size of the input instance to a polylogarithmic one, while keeping the number of processors used at a reasonable level. Problems that can be solved in polylogarithmic time by a multiprocessor machine that uses a polynomial number of processors belong to the class **NC** (and trivially $\mathbf{NC} \subseteq \mathbf{P}$).

Just as there are problems for which it does not seem to be possible to devise polynomial time algorithms, there are also problems that can be solved using a single processor machine in polynomial time, but for which using many processors does not seem to provide a significant advantage over having just one. These problems, that appear to possess an inherent sequentiality, are known as **P**-complete problems. As in the case of **P** and **NP**, it is widely believed that the classes **P** and **NC** differ, but so far a proof has not been found. We will give precise definitions of the concepts introduced in this section in the next chapter.

1.3 The thesis

In this thesis we will introduce two generic greedy algorithms, respectively called GREEDY(π) and MaxDegree(π), and we will consider the complexity of problems that involve their use to solve graph theoretical questions.

We are strongly motivated by results obtained by Satoru Miyano in the paper *The lexicographically first maximal subgraph problems:* **P***-completeness* *and* **NC** *algorithms* [29], and by the results obtained by Pavol Hell and Jaroslav Nešetřil in the paper *On the complexity of H-colouring* [23]; and, often using the same style of proof seen in these two papers, we will obtain results that parallel their results but in different complexity classes. Of course, this does not mean that our results trivially follow from these papers; our proofs are combinatorially more complex and require different structures due to the different complexity classes involved.

The thesis is structured as follows. In Chapter 2 we will introduce the terminology and give the definitions that will be used in the rest of the thesis; in Chapter 3 we will exhibit a whole class of **NP**-complete problems (based around our algorithm GREEDY(π)); and in Chapter 4 we will examine the boundaries between **P** and **NP** for some specific problems, following the style used by Miyano in [29]. In Chapters 5 and 6 we will show that the proof used by Hell and Nešetřil in [23] can be modified to obtain other dichotomy results involving the classes **NP** and $\mathbf{\Sigma_2^p}$, and **P** and $\mathbf{\Delta_2^p}$, respectively. Finally, in Chapter 7 we will introduce and examine the greedy algorithm MaxDegree(π), and study the complexity of related problems.

Chapter 2

Definitions

2.1 Introduction

In this chapter we will define the terminology that will be subsequently used in the document. We will start with some definitions related to complexity theory and then continue with some graph theoretical concepts needed later in the thesis. We will then consider the notion of a greedy algorithm before concluding the chapter by introducing and explaining the generic algorithm GREEDY(π), on which most of this thesis is focused.

2.2 Complexity theory

In this section we will give a brief introduction to complexity theory: for any concept used in the thesis but not defined here we refer the reader to

[6, 15, 31, 39].

A *function* is a string relation of arity 2 in which each string $x \in \{0,1\}^*$ is the first element of exactly one pair. In this thesis we will only be concerned with *decision problems* (as opposed to function problems); that is, with functions in which the only possible solution to each instance is 1 or 0. A *language* is any subset of $\{0,1\}^*$. Given a language L, the corresponding decision problem R_L is: $\{(x,1) : x \in L\} \cup \{(x,0) : x \notin L\}$. Given a decision problem R, we will refer to $\{x \in \{0,1\}^* : (x,1) \in R\}$ as the set of yes-instances, and to $\{x \in \{0,1\}^* : (x,0) \in R\}$ as the set of no-instances of the problem. If L is a language then the corresponding *complementary language* is co-$L = \{0,1\}^* - L$ [39].

Fundamental to complexity theory is the concept of a *complexity class*. A complexity class consists of all problems of a certain kind, decision or function problems for example, that can be solved using a particular model of computation using only a limited quantity of resources. The resources that most researchers tend to be interested in are space and time, and we will take as our model of computation the classical Turing machine (see for example [15, 31]), and, in the case of the class **NC**, the PRAM (see [16, 19]). The complexity classes that we will consider are: **L**, **NC**, **P**, **NP**, $\mathbf{\Delta_2^p}$, $\mathbf{\Sigma_2^p}$ and $\mathbf{\Pi_2^p}$. For any Turing machine M and any language L, we say that M *accepts* L if, given as input any string x: if $x \in L$ then there is an accepting computation of M on input x; if $x \notin L$ then there is no accepting computation of M on input x.

- Complexity class **L** is the class of languages that can be accepted by a deterministic Turing machine in logarithmic space.

- Complexity class **NC** is the class of all languages that are accepted in polylogarithmic time by a PRAM that uses a polynomial number of processors.

- Complexity class **P** is the class of languages that can be accepted by a deterministic Turing machine in polynomial time.

- Complexity class **NP** is the class of languages that can be accepted by a nondeterministic Turing machine in polynomial time.

In order to define the last 3 complexity classes, that is, $\mathbf{\Delta_2^p}, \mathbf{\Sigma_2^p}$ and $\mathbf{\Pi_2^p}$, we need to define *oracle Turing machines* first.

An oracle Turing machine $M^?$ is a multi-tape Turing machine (deterministic or nondeterministic) with a special write-only tape called the *query tape*, and three special states $q_?$ (the *query state*) and q_Y, q_N (the *answer states*). If we take A to be an arbitrary language, the computation of oracle machine $M^?$ with oracle A proceeds like an ordinary Turing machine, except that when a symbol is written on the query tape, the head moves one cell to the right. Depending on whether the current string on the query tape is in A or not, $M^?$ moves from the query state to q_Y or q_N, respectively, and the contents of the query tape are erased (the head moves back to the start of the tape). The answer states allow the machine to behave differently according to the answer obtained from the oracle. We will denote as M^A a Turing machine such that A is its oracle.

We can now define the *polynomial time hierarchy* [37]. Let A be a language. $\mathbf{P}^A$ (resp. $\mathbf{NP}^A$) is the class of languages accepted by M^A where M is a deterministic (resp. nondeterministic) oracle Turing machine which operates in time $p(n)$ for some polynomial $p(n)$.

For a class of languages $\mathcal{C}$,

$$\mathbf{P}^{\mathcal{C}} = \bigcup_{A \in \mathcal{C}} \mathbf{P}^A, \qquad \mathbf{NP}^{\mathcal{C}} = \bigcup_{A \in \mathcal{C}} \mathbf{NP}^A.$$

The polynomial time hierarchy is $\{\mathbf{\Sigma_k^p}, \mathbf{\Pi_k^p}, \mathbf{\Delta_k^p} : k \geq 0\}$, where

$$\mathbf{\Sigma_0^p} = \mathbf{\Pi_0^p} = \mathbf{\Delta_0^p} = \mathbf{P}$$

and for any $k \geq 0$

$$\mathbf{\Sigma_{k+1}^p} = \mathbf{NP}^{\mathbf{\Sigma_k^p}},$$

$$\mathbf{\Pi_{k+1}^p} = \mathbf{co\text{-}NP}^{\mathbf{\Sigma_k^p}},$$

$$\mathbf{\Delta_{k+1}^p} = \mathbf{P}^{\mathbf{\Sigma_k^p}}.$$

As previously mentioned, we will not go beyond the second level of the polynomial time hierarchy, and will therefore only consider problems in $\mathbf{\Delta_2^p} = \mathbf{P^{NP}}$, $\mathbf{\Sigma_2^p} = \mathbf{NP^{NP}}$ and $\mathbf{\Pi_2^p} = \mathbf{co\text{-}NP^{NP}}$.

Given two decision problems X and Y, a *reduction* from X to Y is a function $f : \{0,1\}^* \to \{0,1\}^*$ such that any instance x of X is a yes-instance if, and only if, $f(x)$ is a yes-instance of Y. Of course, it is necessary to consider the resources used to derive $f(x)$ from x: in this thesis we will deal with reductions that can be performed in deterministic logarithmic space. Two decision problems X and Y are *logspace-equivalent* if X is reducible to Y,

Y is reducible to X, and each reduction can be performed using deterministic logarithmic space.

Given a complexity class $\mathcal{C}$ and a language L in $\mathcal{C}$, we will say that L is complete for $\mathcal{C}$ (or *$\mathcal{C}$-complete*) if any language $L' \in \mathcal{C}$ can be reduced to it. If any language in $\mathcal{C}$ is reducible to L, but L is not known to be in $\mathcal{C}$, then the problem is *$\mathcal{C}$-hard.*

All the problems that we will consider are related to graph theory, and we will introduce the relevant definitions in the next section.

2.3 Graph theoretical definitions

In this section we will define most of the graph theoretical concepts that will be used in the rest of the document. For any concepts used in the thesis but not defined here, or for an introduction to graph theory, see [4, 20, 21].

A *graph* $G = (V, E)$ is a mathematical structure consisting of a finite set V of elements called *vertices* (or *nodes*), and a set E of unordered pairs of distinct vertices from V, called *edges. Adjacent vertices* are two vertices that are joined by an edge. If vertex v is an endpoint of edge e then v is said to be *incident* on e, and e is *incident* on v. The *degree* of a vertex v in a graph G, denoted $deg(v)_G$ (or $deg(v)$ if the graph G is understood), is the number of edges incident on v. The maximum degree of a graph G is denoted by $\Delta(G)$.

A *directed graph* (or *digraph*) is such that the edges are ordered pairs of distinct vertices in V. If $e = (x, y)$ is an edge of a digraph then x is the *initial endpoint* of e and y is the *terminal endpoint* of e. If there is an edge from a vertex x to a vertex y, but not one from y to x, then we will say that y is a *child* of x, and x is the *predecessor* (or *parent*) of y. When referring to digraphs, the *out-degree* of a vertex x is the number of edges that have x as their initial endpoint, and the *in-degree* is the number of edges that have x as their terminal endpoint. The *underlying graph* of a digraph G is the graph obtained by replacing each edge of G by the corresponding undirected edge.

The *order* of a graph $G = (V, E)$ is given by the number of vertices, and it is denoted by $|G|$ or $|V|$. A *path* is a sequence of edges of the form $(v_0, v_1), (v_1, v_2), \ldots, (v_{n-1}, v_n)$. A *cycle* is a path in which $v_0 = v_n$ and all vertices on the path are distinct. Given a subset U of the vertices of a graph (or digraph) $G = (V, E)$, the subgraph of G *induced by* U, $\langle U \rangle_G$, is: $\langle U \rangle_G = (U, E_U)$, where $E_U = \{(x, y) \in E | x, y \in U\}$. These definitions hold for both graphs and digraphs.

In this thesis we will mainly deal with properties on graphs (only discussing properties on digraphs in one theorem). Let π be some property of graphs. In the following chapters we will often say that a graph is (or possesses property) π. We will now define the properties π that we will consider.

- A graph has *bounded degree*, where the bound is k, if every vertex has degree at most k.
- A graph is *acyclic* if it does not contain any cycle. It is *k-cycle free* if it does not contain a cycle of length k.
- A *clique* (or complete graph) with n vertices (denoted K_n) is a graph

with n vertices in which each vertex is adjacent to all the others.

- A graph is *planar* if it can be drawn on the plane without any two edges crossing.

- A planar graph is *outerplanar* if it can be drawn on the plane with all its vertices on the same face, which is generally chosen to be the exterior face.

- A graph is *bipartite* if its vertices can be partitioned into two disjoint subsets U_1 and U_2 such that each edge joins a vertex from U_1 to one from U_2. A *complete bipartite* graph has edge set $E = \{(u, v) | u \in U_1, v \in U_2\}$, and we will denote it as K_{n_1,n_2}, where n_i is the number of vertices in U_i. The complete bipartite graph $K_{1,n}$ is called a *star* of size n, or n-star: we will refer to the single vertex in U_1 as the *centre* of the star, and to the vertices in U_2 as the *leaves*.

- Given a graph $G = (V, E)$, the corresponding edge graph $L(G) = (E, D)$ is the graph that has as vertex set the edge set of G, and such that 2 vertices in $L(G)$ are adjacent if, and only if, the corresponding edges in G have a vertex in common. We say that a graph G is an *edge graph* if there exists a graph H such that G is isomorphic to the edge graph $L(H)$ of H.

- An *interval graph* is a graph such that there exists a set of intervals on the real line in a one-to-one correspondence with the vertices of the graph. Two vertices are adjacent if, and only if, their corresponding intervals intersect.

- Given a cycle C in a graph G, a *chord* of C is an edge of G joining two vertices of C which are not adjacent in the cycle. A graph G is called *chordal* if every cycle in G of length ≥ 4 has a chord.

- Let G and H be graphs. A *homomorphism from G to H* is a map f from the vertices of G to the vertices of H such that if (u, v) is an edge of G then $(f(u), f(v))$ is an edge of H. It is an *isomorphism* if f is also onto, one-to-one and the inverse map is a homomorphism. The *H-colouring problem* is the problem whose instances are graphs G and whose yes-instances are those graphs G for which there is a homomorphism from G to H. We will refer to a graph G that possesses such a property as being *H-colourable.*

- A graph is 3-*colourable* if each vertex can be coloured with a unique colour from red, white and blue so that two adjacent vertices are coloured differently; and the 3-*colouring problem* has as an instance a graph G and as a yes-instance a graph G that is 3-colourable.

When talking about graph theoretical properties, we will often say that a property π is *hereditary*: by this we mean that whenever we have a graph with the property π, the deletion of any vertex and its incident edges does not produce a graph violating π, *i.e.*, π is preserved by induced subgraphs. A property π is *non-trivial* on a class of graphs $\mathcal{C}$ if there are infinitely many graphs from $\mathcal{C}$ satisfying π and infinitely many violating it. Note that the definition of hereditary and non-trivial property is valid for both graphs and digraphs.

In the following chapters, we will use graph theoretical properties in the context of greedy algorithms. We will discuss greedy algorithms in the next section.

2.4 Greedy algorithms

On page 6, we showed a greedy algorithm that finds a maximal independent set of vertices. The algorithm examines the vertices of a graph following some particular order and chooses and rejects vertices one at a time. This procedure will also be used in the algorithms defined later in the thesis. In this section we will show that the strategy used by the algorithm can be generalised and applied to structures different from graphs.

An *independence system* is a pair $I = (E, \mathcal{F})$, where E is a finite ordered set of elements $\{e_1, \ldots, e_n\}$, and $\mathcal{F}$ is a family of subsets of E; each element of $\mathcal{F}$ is called an *independent set.* In addition we require that independent sets have the property that $\emptyset \in \mathcal{F}$, and that independence is hereditary , that is, if a set is in $\mathcal{F}$, then so are all its subsets. The related problem consists of greedily finding the independent set $G = \{e_{j_1}, \ldots, e_{j_k}\}$ for I, where

- $1 = j_0 \leq j_1 < j_2 < \ldots < j_k < j_{k+1} = n + 1$
- for all $j_i < l < j_{i+1}$, $G_i \cup \{e_l\}$ is not independent.

The problem can be solved by the following generic greedy algorithm.

Generic Greedy Maximal Independent Set algorithm

begin

$G := \emptyset$

for $i := 1$ **to** n

if $G \cup \{e_i\} \in \mathcal{F}$ **then** $G := G \cup \{e_i\}$

end

It is clear that the algorithm proceeds as the one on page 6, but instead of examining the vertices of a graph, it considers the elements of an ordered set. In this thesis we will only deal with graph theoretical properties, and we will not therefore pursue this topic any further, but more details on the subject can be found, for example, in [19].

2.5 GREEDY(π)

Let $G = (V, E)$ be a graph (directed or undirected) and suppose that the vertices of V are linearly ordered. Given a subset $S = \{s_0, s_1, s_2, \ldots, s_k\}$ of V, where the induced ordering is $s_0 < s_1 < \ldots < s_k$, we can define a *lexicographic order* on the set of all subsets of S as follows (we call it lexicographic because we consider $s_0, s_1, \ldots, s_k$ to be our alphabet):

- for subsets $U = \{u_1, u_2, \ldots, u_p\}$ and $W = \{w_1, w_2, \ldots, w_k\}$ of S, where $u_1 < u_2 < \ldots < u_p$ and $w_1 < w_2 < \ldots < w_k$, we say that U is *lexicographically smaller than* W if:

- there is a number t, where $1 \leq t \leq p$, such that $u_t < w_t$ and $u_i = w_i$, for all i such that $1 \leq i < t$; or
- $k > p$ and $u_i = w_i$, for all i such that $1 \leq i \leq p$.

Miyano [29] proved that the problem of computing the lexicographically first maximal subgraph of a given graph, where this subgraph should satisfy some fixed polynomial time testable, non-trivial, hereditary property π, is **P**-hard (even when the given graph is restricted to be either bipartite or planar and π is non-trivial on the class of bipartite or planar graphs, respectively). Because of the stipulations on π, the lexicographically first maximal subgraph satisfying the property π can be computed by a generic greedy algorithm. Note that Miyano's result is widely applicable; to *any* polynomial time testable, non-trivial, hereditary property π, such as whether a graph is planar, bipartite, acyclic, of bounded degree, an interval graph, chordal, and so on. Miyano states that his work was inspired by that of Asano and Hirata [2], Lewis and Yannakakis [27], Watanabe, Ae and Nakamura [40] and Yannakakis [42] on node- and edge-deletion problems in **NP**. Typical in this work is the result of Lewis and Yannakakis [27] that the problem of finding the minimum number of nodes needing to be deleted from a graph so that the graph satisfies a fixed polynomial time testable, non-trivial, hereditary property π is **NP**-hard.

Of course, a tacit assumption in Miyano's work is that the vertices of any graph are linearly ordered. In the following chapters, inspired by Miyano's results, we consider computing lexicographically first maximal subgraphs of given graphs, where these subgraphs should satisfy some given non-trivial, hereditary property π, except that now the graphs come equipped with not just one linear ordering of their vertices but several. Hence, for a given graph we will be involved with a collection of lexicographically first maximal subgraphs and not just one. Note that if we gave our linear orderings explicitly then a graph on n vertices could only come with a polynomial (in n) number of such linear orderings (as otherwise it would be unreasonable to define that the whole input has size n). In order to work with an exponential number of linear orderings, we present our collection of linear orderings in the form of a partial order, *i.e.*, an acyclic digraph, with a source vertex providing the (common) least element of any of the linear orderings. Let s be our source vertex. We think of a partial ordering P as encoding a collection of linear orderings of the form $s = s_0 < s_1 < s_2 < \ldots < s_k$, where s_{j+1} is a child of s_j, for $0 \leq j < k$, and s_k has no children. Note that we will always assume that every vertex $v \in V$ is reachable from s in P, that is, there is at least one path starting from s that contains vertex v.

Similarly to Miyano's deterministic scenario, we will use a nondeterministic polynomial time greedy algorithm that computes all lexicographically first maximal subgraphs. Our algorithm, called GREEDY(π), takes as input 3 arguments: a graph (directed or undirected) $G = (V, E)$, a directed graph $P = (V, D)$ and a specified vertex $s \in V$. The algorithm GREEDY(π) is as follows:

```
input(G,P,s)
   S := ∅
   current-vertex := s
```

```
  if π(S∪{current-vertex},G) then                         (*)
    S := S∪{current-vertex}
    fi
  while current-vertex has at least one child in P do
    current-vertex := a child of current-vertex in P
    if π(S∪{current-vertex},G) then                       (**)
      S := S∪{current-vertex}
      fi
    od
output(S)
```

where $\pi(S \cup \{current\text{-}vertex\}, G)$ is a predicate evaluating to 'true' if, and only if, the subgraph of G induced by the vertices of $S \cup \{current\text{-}vertex\}$ satisfies π. We say that a vertex v is the *current-vertex* if we have 'frozen' an execution of the algorithm GREEDY(π) immediately prior to executing either line $(*)$ or line $(**)$ and the value of the variable *current-vertex* at this point is v.

We will now explain why GREEDY(π) fits the description of the greedy procedure detailed in Section 2.4. Given as instance a graph G and a partial ordering P, the algorithm selects a path in the ordering and then proceeds to examine the vertices on the path following a linear order. Once the choice of the path has been made, only the subgraph of G induced by the vertices on the path is considered, and such vertices are examined one at a time like in the case of the outlined procedure. The algorithm chooses the vertices during the execution, instead of making the choice before and then operating according to the linear order, but it is straightforward to see that this does not make a difference.

The algorithm GREEDY(π) is very general, and its behaviour changes according to the structure of the graph P which from now on will be called the *ordering*. If the ordering consists of a simple path from vertex s then GREEDY(π) becomes a deterministic algorithm, because at any stage in the execution of the algorithm *current-vertex* has at most one child, and the order in which vertices are examined is dictated by their distance from vertex s. The output of the algorithm is, in this case, always the same after every execution, and it consists of a set of vertices.

If the ordering, that is, the graph P, is not a simple path, but is such that there are two or more different paths starting from vertex s, then the algorithm becomes nondeterministic and produces a collection of sets of vertices as outputs. Note that we will only consider orderings P that contain no cycles, but the reader is invited to check that the complexity of the problem does not change even if we allow P to contain cycles. A slightly modified version of the algorithm could remember which vertices have been already examined, therefore always choosing a child of the current vertex that has not been considered before, and otherwise proceed exactly as GREEDY(π). Since our acyclic approach is a subproblem of the general digraph problem, our results will hold in both cases. We have not found any results regarding the complexity of the problem involving linear extensions of partial orderings.

Let $\mathcal{C}$ be a class of graphs and let π be some property of graphs. The problem GREEDY(ordering, $\mathcal{C}$, π) has: as its instances tuples (G, P, s, t), where G is a graph from $\mathcal{C}$, P is an ordering of the vertices of G and s and t are vertices of G; and as its yes-instances those instances for which there exists an execution of the algorithm GREEDY(π) on input (G, P, s) resulting in the vertex t being output.

In the following chapters we will discuss the complexity of the problem GREEDY(ordering, $\mathcal{C}$, π) for different values assigned to the parameters ordering, $\mathcal{C}$ and π. We will begin, in the next chapter, by considering properties π that are testable in polynomial time, hereditary and non-trivial on the chosen class of graphs $\mathcal{C}$.

Chapter 3

A class of NP-complete problems

3.1 Introduction

The decision version of the problem lexicographically first maximal subgraph satisfying property π, LFSMP(π), discussed by Miyano in [29], and mentioned in the previous section, can be restated as GREEDY(linear ordering, $\mathcal{C}, \pi$); and therefore Miyano's main result from [29] can be stated as follows.

Theorem 3.1 *Let π be a polynomial time testable, non-trivial, hereditary property on the class of graphs $\mathcal{C}$, where $\mathcal{C}$ is the class of all graphs, the class of planar graphs or the class of bipartite graphs. Then the problem GREEDY(linear ordering, $\mathcal{C}$, π) is* $\mathbf{P}$*-complete.* □

In this chapter we will parallel this result in the class **NP** by showing that the problem GREEDY(partial ordering, $\mathcal{C}$, π) is **NP**-complete for any property π which is hereditary, testable in polynomial time and non-trivial on $\mathcal{C}$, where $\mathcal{C}$ is the class of all graphs or the class of planar bipartite graphs.

We will start our discussion by considering the conditions that we need to apply to our ordering to move from the setting of **P** to the setting of **NP**. We will prove that the problem GREEDY(partial ordering, planar bipartite, independent set) is **NP**-complete and we will use this result as a base to prove that for any property π that is testable in polynomial time, hereditary and non-trivial, the problem GREEDY(partial ordering, undirected graphs, π) is complete for **NP** as well. We will conclude the chapter by showing that the result holds also if we restrict to planar bipartite graphs and if we consider directed graphs. We remark here that many of the results of this chapter appeared in the paper by A. Puricella and I. A. Stewart, A generic greedy algorithm, partially-ordered graphs and NP-completeness, *Proceedings of* 27*th International Workshop on Graph-Theoretic Concepts in Computer Science* (*WG*'01) [33].

3.2 Tree orderings

In this section we will show that if we equip graphs with a *tree ordering*, that is, with a partial ordering of their vertices in the form of a rooted out-tree then the problem GREEDY(tree ordering, $\mathcal{C}$, π) still resides in **P**. Note that this is essentially the same as considering graphs equipped with a polynomial number of linear orderings of their vertices, because in a directed tree there is at most one path between any two given vertices. It is straightforward to see that the problem GREEDY(tree ordering, $\mathcal{C}$, π) is in $\mathbf{P}$, because a deterministic algorithm can be used to find the unique path between vertices s and t, for any instance (G, T, s, t), and the vertices on such a path can be used as a linear ordering on G. We actually prove here a stronger result.

Proposition 3.2 *Let $\mathcal{C}$ be any class of graphs and let π be any property of graphs. The problems GREEDY(tree ordering, $\mathcal{C}$, π) and GREEDY(linear ordering, $\mathcal{C}$, π) are* **NC***-equivalent.*

Proof Let G be a graph of size n; let T be a tree ordering of the vertices of G, with root s; and let t be some vertex of G. Consider the following **NC** algorithm. Assign a processor to every vertex v in T. In shared-memory cell $M[v]$, register the parent of vertex v in T (if there is one). Consequently, we have hidden away in shared-memory cells $M[1 \ldots n]$ a collection of linked lists, with the root s of the tree as the head of every linked list, representing the paths emanating from the root in T. Perform the usual list-ranking algorithm on these linked lists (see, for example, [16]), and also (as part of the list-ranking process) mark all those vertices which lie on a path between s and t. After list-ranking, we can ascertain the precise path in T from s to t. Hence, we have reduced the problem GREEDY(tree ordering, $\mathcal{C}$, π) to the problem GREEDY(linear ordering, $\mathcal{C}$, π). Our algorithm can easily be implemented in $\mathcal{O}(\log n)$ time using n processors on an EREW PRAM. The result follows. $\square$

The following is now immediate from Miyano's result.

Corollary 3.3 *Let π be a polynomial time testable, non-trivial, hereditary property on the class of graphs $\mathcal{C}$ where $\mathcal{C}$ is the class of all graphs, the class of planar graphs or the class of bipartite graphs. Then the problem GREEDY(tree ordering, $\mathcal{C}$, π) is* **P***-complete.* □

In [29], **NC** algorithms for certain problems of the form GREEDY(linear ordering, $\mathcal{C}$, π) (for specific classes of graphs $\mathcal{C}$ and properties π) were derived. Proposition 3.2 yields **NC** algorithms for these problems when the graphs are equipped with tree orderings of their vertices, as opposed to linear orderings. We therefore obtain (using the results of [29]) the following corollaries.

Corollary 3.4 *The problem GREEDY(tree ordering, graphs with maximum degree* 2*, independent set) is in* **NC**.

Note that graphs with maximum degree 2 simply consist of cycles, paths and independent sets of vertices.

Corollary 3.5 *If we take π to be the property 3-cycle free or the property 4-cycle free we obtain the following result. The problem GREEDY(tree ordering, graphs with maximum degree* 3*, π) is in* **NC**.

3.3 Independent sets

In order to prove our main result of the chapter, in the next section, we need to first establish a completeness result for the problem GREEDY(partial ordering, planar bipartite, independent set). In the following theorem we will in fact prove a stronger result, and show that GREEDY(partial ordering, planar bipartite acyclic of maximum degree 3, independent set) is **NP**-complete. As every acyclic graph is planar and bipartite, the problem could alternatively be stated as GREEDY(partial ordering, acyclic of maximum degree 3, independent set).

Theorem 3.6 *The problem GREEDY(partial ordering, planar bipartite and acyclic of maximal degree 3, independent set) is* **NP***-complete.*

Proof The problem is clearly in **NP** as it can be solved in polynomial time by GREEDY(independent set); to prove completeness we reduce from the known **NP**-complete problem Directed Hamiltonian Path (DHP): whose instances are triples (G, s, t), where G is a digraph and s and t are vertices of G; and whose yes-instances are instances for which there is a Hamiltonian path in G from s to t (see [15]).

Let $(G = (V, E), s, t)$ be an instance of DHP of size n. W.l.o.g. we assume that $|V| > 2$, that the vertex set of G is $\{1, 2, \ldots, n\}$ and that $s = 1$ and $t = n$. Corresponding to this instance, we build an instance (G', P', s', t') of GREEDY(partial ordering, planar bipartite, independent set) (for brevity, we call this problem $\mathcal{H}$). The vertex set V' of G' and P' is

$$\{u_{i,j}, v_{i,j}, w_{i,j}, z_j : i, j = 2, 3, \ldots, n-1\} \cup \{x, s', t'\}.$$

The edges of G' are

$$\{(u_{i,j}, v_{i,j}), (u_{i,j}, w_{i,j}) : i, j = 2, 3, \ldots, n-1\}$$
$$\cup\{(w_{i,j}, u_{i+1,j}) : i = 2, 3, \ldots, n-2; j = 2, 3, \ldots, n-1\}$$

$$\cup\{(w_{n-1,j}, z_j) : j = 2, 3, \ldots, n-1\}$$
$$\cup\{(z_j, w_{n-1,j+1}) : j = 2, 3, \ldots, n-2\}$$
$$\cup\{(z_{n-1}, t')\}$$

and the edges of P' are

$$\{(v_{i,j}, v_{i+1,j'}) : i = 2, 3, \ldots, n-2; j, j' = 2, 3, \ldots, n-1; (j, j') \in E\}$$
$$\cup\{(s', v_{2,j}) : j = 2, 3, \ldots, n-1; (1, j) \in E\}$$
$$\cup\{(v_{n-1,j}, x) : j = 2, 3, \ldots, n-1; (j, n) \in E\}$$
$$\cup\{(x, u_{2,2})\}$$
$$\cup\{(u_{i,j}, w_{i,j}) : i = 2, 3, \ldots, n-2; j = 2, 3, \ldots, n-1\}$$
$$\cup\{(w_{i,j}, u_{i+1,j}) : i = 2, 3, \ldots, n-2; j = 2, 3, \ldots, n-1\}$$
$$\cup\{(u_{n-1,j}, u_{2,j+1} : j = 2, 3, \ldots, n-2\}$$
$$\cup\{(u_{n-1,n-1}, w_{n-1,2})\}$$
$$\cup\{(w_{n-1,j}, z_j) : j = 2, 3, \ldots, n-1\}$$
$$\cup\{(z_j, w_{n-1,j+1}) : j = 2, 3, \ldots, n-2\}$$
$$\cup\{(z_{n-1}, t')\}$$

The construction of the instance (G', P', s', t') is illustrated in Figures 3.1, 3.2 and 3.3 which depict: a digraph G; the resulting graph G'; and the resulting partial ordering P', respectively. Note that the graph G' is always planar and bipartite. Note also that G' depends solely upon n and not on the edges of G; and that the only portion of P' depending upon the edges of G is the initial portion involving the v-vertices (the rest of P' is a linear ordering).

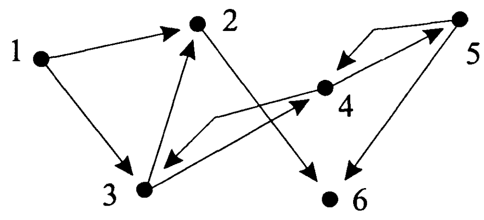


Figure 3.1: A digraph G.

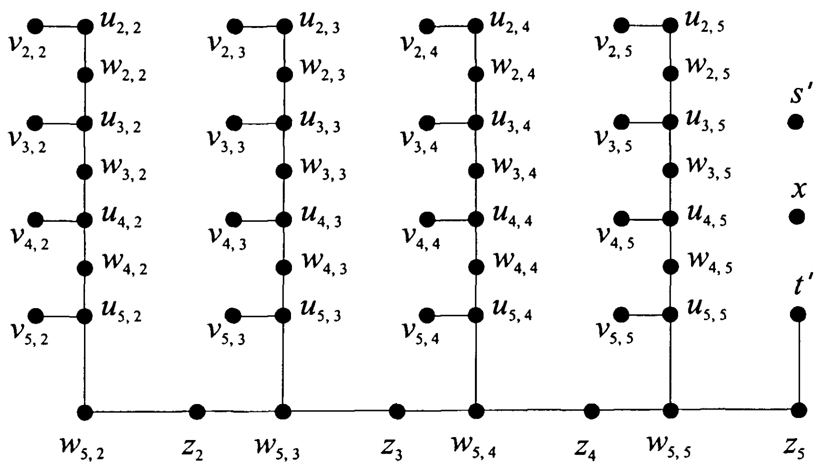


Figure 3.2: The graph G' corresponding to G.

Suppose that (G, s, t) is a yes-instance of DHP. Then there is a Hamiltonian path $s = s_1, s_2, s_3, \ldots, s_{n-1}, s_n = t$ in G. Consider the following path in P':

$$s', v_{2,s_2}, v_{3,s_3}, \ldots, v_{n-1,s_{n-1}}, x$$

(note that this is indeed a path in P'). In the execution of the algorithm GREEDY(independent set) on (G', P', s', t'), following this path in P' clearly

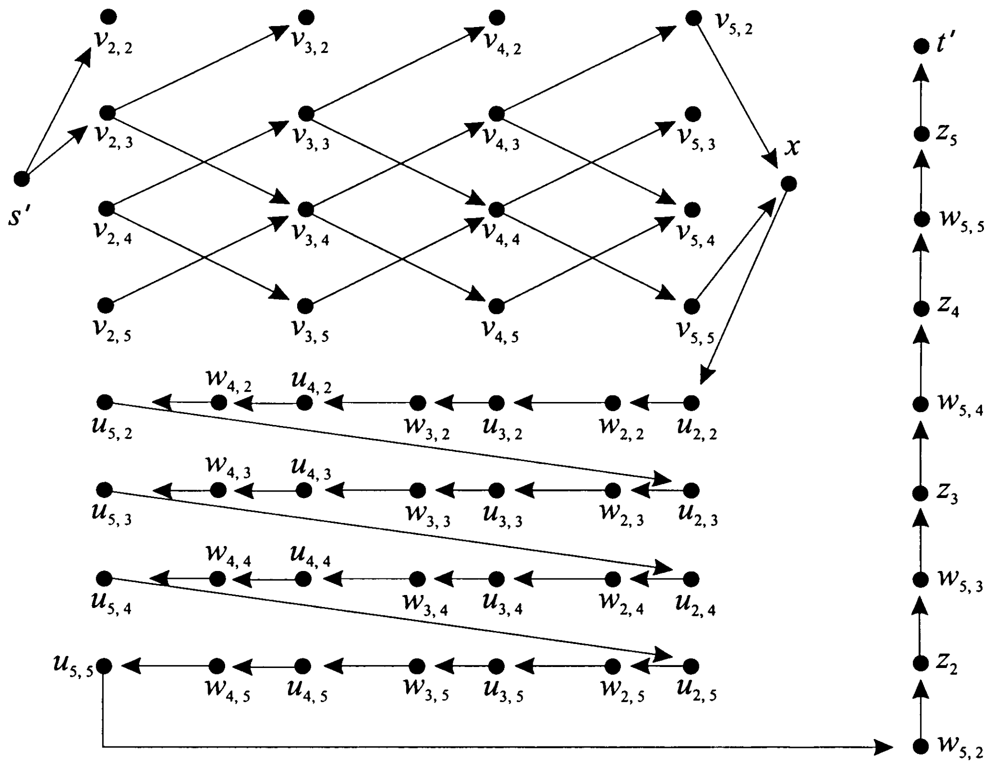


Figure 3.3: The partial ordering P' corresponding to G.

results in the vertices of $\{s', v_{2,s_2}, v_{3,s_3}, \ldots, v_{n-1,s_{n-1}}, x\}$ all being output.

Henceforth, the path chosen in P' is fixed. With reference to Figure 3.2, following this path we work down the first column of u- and w-vertices of G' (that is, the column with index 2, *i.e.*, involving vertices of the form $u_{-,2}$ and $w_{-,2}$) then the second column (the column with index 3), until having worked down the last column (the column with index $n-1$), we work along the bottom row of w- and z-vertices. For every $j = 2, 3, \ldots, n-1$, a vertex $v_{i,j}$, for some i, has been output by the algorithm GREEDY(independent set); that is, there is exactly one v-vertex output from every column. Hence, as we work down the columns of u- and w-vertices, the vertex $w_{n-2,j}$ is output by the algorithm GREEDY(independent set) but the vertex $u_{n-1,j}$ is not, for all $j = 2, 3, \ldots, n-1$. Consequently, when we work along the bottom row of w- and z-vertices of G', the vertex $w_{n-1,j}$ is output but the vertex z_j is not, for all $j = 2, 3, \ldots, n-1$. Finally, the vertex t' is output. Hence, (G', P', s', t') is a yes-instance of $\mathcal{H}$.

Conversely, suppose that (G', P', s', t') is a yes-instance of $\mathcal{H}$ and consider an execution of the algorithm GREEDY(independent set) witnessing this fact. The path chosen in P' from s' to x yields a path of length $n-1$ in G from 1 to n. Suppose that this path in G is such that a vertex j appears on it more than once. This means that vertices $v_{i,j}$ and $v_{i',j}$ appear on the path in P' from s' to x, where $i \neq i'$. Hence, with reference to Figure 3.2, there must be some column in G' for which a v-vertex has not been output by the algorithm GREEDY(independent set). Let the largest index of any such column be k. When we work down the u- and w-vertices of column k in G' in our execution of the algorithm GREEDY(independent set), the result is that all of the u-vertices are output and none of the w-vertices are. When we work down the u- and w-vertices of column m, for any $m > k$, in our execution of the algorithm GREEDY(independent set), the result is that the vertex $u_{n-1,m}$ is not output. Hence, when we work along the bottom row of w- and z-vertices in our execution of the algorithm GREEDY(independent set), the vertices $z_k, z_{k+1}, \ldots, z_{n-1}$ are all output but not the vertex t'. This yields a contradiction; and so we have a Hamiltonian path in G from 1 to n. Hence, (G, s, t) is a yes-instance of DHP.

As the construction (G', P', s', t') from (G, s, t) can clearly be completed using logspace, the result follows. $\square$

3.4 Polynomial time hereditary properties

In this section, we consider the problem GREEDY(partial ordering, planar bipartite, π) where π is a polynomial time testable, non-trivial, hereditary property. We begin with some graph-theoretic definitions specific to the proofs in this section.

We refer to a set of disjoint edges as *independent edges.* A *cut-point* of a connected graph G is a vertex c such that its removal (along with its incident edges) from G results in a graph with at least 2 connected components. A *component relative to a cut-point* c is a subgraph consisting of c, one of the derived connected components and all those edges of G joining c and a vertex of the component. If a connected graph does not have any cut-points then it is *biconnected.* Later in the proof, in order to decide whether a graph possesses a property π, we will need to examine the size of the connected components relative to a cut-point. The following definitions will be used for this purpose.

Let $\mathbf{a} = (a_1, a_2, \ldots, a_s)$ and $\mathbf{b} = (b_1, b_2, \ldots, b_t)$ be two tuples of positive integers. We order these tuples lexicographically as follows. We say that $\mathbf{a} >_L \mathbf{b}$ if either:

- there exists some $i \in \{1, 2, \ldots, \min\{s, t\}\}$ such that $a_j = b_j$, for all $j \in \{1, 2, \ldots, i-1\}$, and $a_i > b_i$; or

- $s > t$ and $a_j = b_j$, for all $j \in \{1, 2, \ldots, t\}$.

The *α-sequence* α_G of a connected graph G is defined as follows. Suppose that G is not biconnected. If c is a cut-point of G whose removal results in a graph with k connected components then define $\alpha_{c,G} = (n_1, n_2, \ldots, n_k)$, where $n_1 \geq n_2 \geq \ldots \geq n_k$ are the numbers of vertices in the components relative to c. We define α_G to be the lexicographically-minimal tuple of the (non-empty) set $\{\alpha_{c,G} : c \text{ is a cut-point of } G\}$, and we define c_G to be any cut-point for which $\alpha_G = \alpha_{c_G,G}$. If G is biconnected then it does not contain any cut-points; therefore the number of vertices in any connected component relative to a vertex has size $|G|$. We then define $\alpha_G = (|G|)$ and c_G as any vertex.

Given a graph G with connected components $G_1, G_2, \ldots, G_k$, the *β-sequence* β_G of G is defined as $(\alpha_{G_1}, \alpha_{G_2}, \ldots, \alpha_{G_k})$, where $\alpha_{G_1} \geq_L \alpha_{G_2} \geq_L \ldots \geq_L \alpha_{G_k}$. A β-sequence is therefore a tuple of tuples of integers.

Now for the main result of this section.

Theorem 3.7 *Let π be a property satisfying the following conditions:*

(i) π is non-trivial on planar bipartite graphs;

(ii) π is hereditary on induced subgraphs;

(iii) π is satisfied by all sets of independent edges; and

(iv) π is polynomial time testable.

Then the problem GREEDY(partial ordering, planar bipartite, π) is complete for **NP**.

Proof For brevity, we refer to the problem GREEDY(partial ordering, planar bipartite, π) as $\mathcal{G}$. The property π is, by assumption, non-trivial on planar bipartite graphs. It follows that amongst all planar bipartite graphs violating π, there must be (at least) one with smallest β-sequence, where β-sequences are ordered lexicographically and where the comparison of components, *i.e.*, α-sequences, is according to $\geq_L$. Let us call such a graph J; that is,

$$\beta_J = \min\{\beta_G : G \text{ is a planar bipartite graph violating } \pi\}.$$

Let $J_1, J_2, \ldots, J_k$ be the connected components of J ordered according to $\alpha_{J_1} \geq_L \alpha_{J_2} \geq_L \ldots \geq_L \alpha_{J_k}$. It follows that J has β-sequence $\beta_J = (\alpha_{J_1}, \alpha_{J_2}, \ldots, \alpha_{J_k})$. Let $c = c_{J_1}$ and let the connected components of J_1 relative to c be $I_0 \cup \{c\}, I_1 \cup \{c\}, \ldots, I_m \cup \{c\}$, where $|I_0| \geq |I_1| \geq \ldots \geq |I_m|$. Denote by I_* the subgraph of J_1 induced by the vertices of $I_1 \cup \ldots \cup I_m$. By (*ii*) and (*iii*) it follows that π is satisfied by any independent set of vertices, and so $I_0 \cup \{c\}$ must contain at least one edge (otherwise J would be a set of independent vertices).

To prove the **NP**-completeness of the problem $\mathcal{G}$, we reduce from the problem GREEDY(partial ordering, planar bipartite, independent set), which, for brevity, we denote by $\mathcal{H}$, and which was proven to be **NP**-complete in Theorem 3.6. That is, from an instance (G, P, s, t) of $\mathcal{H}$, we create an instance (G', P', s', t') of $\mathcal{G}$ (with the appropriate properties).

We will divide the construction of G' from G into three phases. For any subset of vertices U of J, we denote by $\langle U \rangle$ the subgraph of J induced by the vertices of U. Note that as $\langle I_0 \cup \{c\} \rangle$ contains at least one edge and is connected, there exists a vertex d of $I_0 \cup \{c\}$ such that (c, d) is an edge of $\langle I_0 \cup \{c\}\rangle$.

Phase 1 For each vertex u of G, we attach a copy of $\langle I_* \cup \{c\}\rangle$ by identifying u with c (all such copies are disjoint). Call the resulting graph $\tilde{G}$. Note that the vertex set of $\tilde{G}$ consists of the vertices of G, which we call the *G-vertices*, together with disjoint copies of the vertices of I_*. As both $\langle I_*\rangle$ and G are planar and bipartite, $\tilde{G}$ maintains these properties.

Phase 2 We replace each edge (u, v) of $\tilde{G}$, where u and v are G-vertices, by a copy of $\langle I_0 \cup \{c\}\rangle$ by identifying u with c and v with d (all such copies are disjoint). Note that our choice of d results in the graph so formed being planar and bipartite.

Phase 3 We add disjoint copies of $J_2, J_3, \ldots, J_k$ to obtain G', which is clearly planar and bipartite.

The partial ordering P' consists of a linear ordering onto which is concatenated the partial ordering P (of the G-vertices). The linear ordering consists of: all vertices of G' that are vertices of some copy of $\langle I_0 \setminus \{d\}\rangle$; followed by all vertices in the copies of $\langle I_*\rangle$; followed by all vertices of $J_2, J_3, \ldots, J_k$. It does not matter how we order the vertices of some copy of $\langle I_*\rangle$, for example, in the linear ordering. We concatenate this linear ordering prior to P by including an edge from the last vertex of the linear ordering to the vertex s of P. Denote the vertex s' to be the first vertex of the above linear ordering, and denote the vertex t' to be the G-vertex of G' formerly known as t. Our construction can be visualised in Figure 3.4.

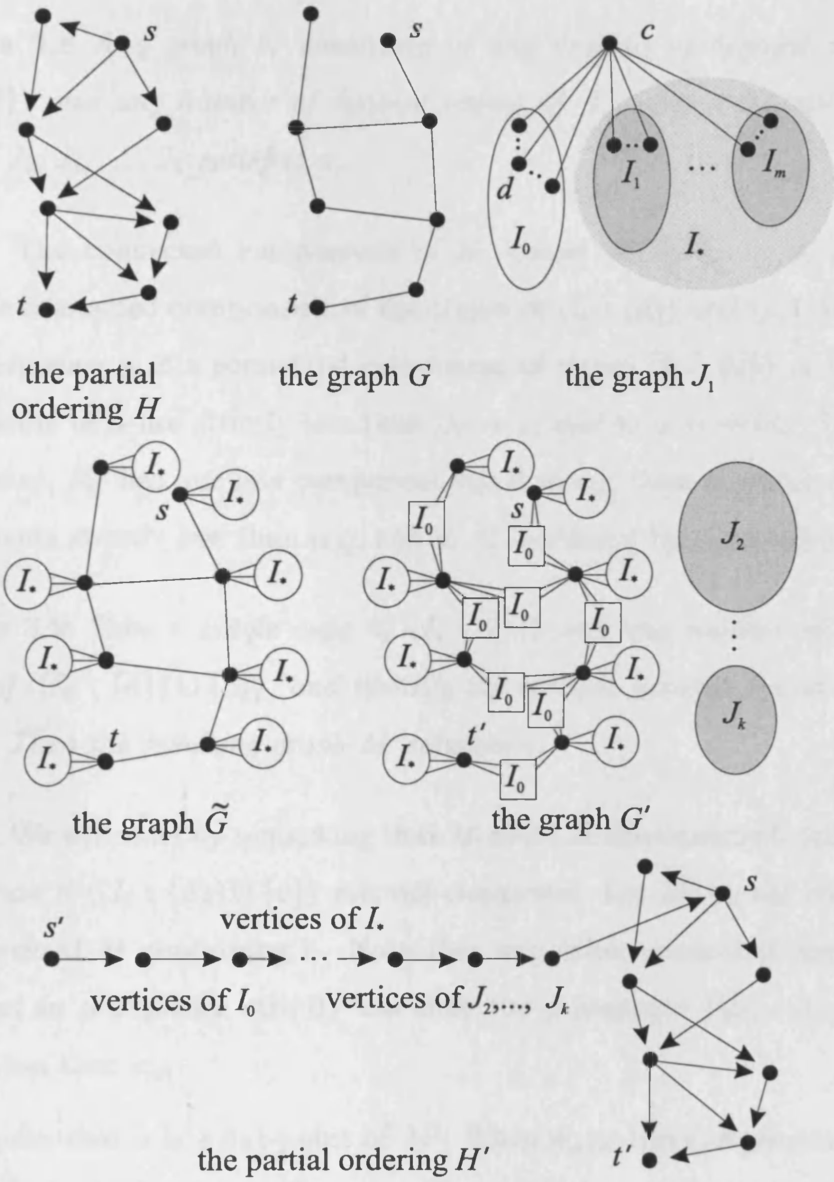


Figure 3.4: Our basic construction.

We will now prove three lemmas to be used in the remainder of the proof.

Lemma 3.8 *Any graph K consisting of any number of disjoint copies of $\langle I_0 \setminus \{d\}\rangle$ plus any number of disjoint copies of $\langle I_*\rangle$ plus a disjoint copy of each of $J_2, J_3, \ldots, J_k$ satisfies π.*

Proof The connected components of K consist of $J_2, J_3, \ldots, J_t$ together with the connected components of the copies of $\langle I_0 \setminus \{d\}\rangle$ and $\langle I_*\rangle$. Consider the α-sequence α of a connected component of either $\langle I_0 \setminus \{d\}\rangle$ or $\langle I_*\rangle$. All components of α are strictly less than $|I_0| + 1$; and so α is strictly less than α_{J_1}. Hence, β_K has one less component equal to α_{J_1} than β_J, with all other components strictly less than α_{J_1}; and so K satisfies π by minimality of β_J.□

Lemma 3.9 *Take a single copy of $\langle I_* \cup \{c\}\rangle$ and any number of disjoint copies of $\langle (I_0 \setminus \{d\}) \cup \{c\}\rangle$, and identify the vertices named c in all of these graphs. Then the resulting graph M satisfies π.*

Proof We will start by remarking that M could be disconnected: this would be the case if $\langle (I_0 \setminus \{d\}) \cup \{c\}\rangle$ was not connected. Let M' be the connected component of M containing c. Note that any other connected component of M has an α-sequence strictly less than the α-sequence $(|I_0| + 1)$; and so strictly less than α_{J_1}.

Suppose that c is a cut-point of M'. Then $\alpha_{c,M'}$ has components $|I_1| + 1, |I_2| + 1, \ldots, |I_m| + 1$ as well as possibly some other components which are all strictly less than $|I_0| + 1$. Hence, by arguing as in the proof of Lemma 3.8, $\alpha_{M'}$ is strictly less than α_{J_1}. By the remark above, β_M is strictly less than β_J and so M satisfies π by the minimality of β_J.

Suppose that c is not a cut-point of M'. Then $I_* = I_1$, *i.e.*, $m = 1$, and $M' = \langle I_* \rangle$; hence, $\alpha_{M'}$ is at most $(|I_1| + 1)$. Any connected component of M different from M' has size at most $|I_0| - 2$, and so $\alpha_{J_1} = (|I_0| + 1, |I_1| + 1)$ is strictly greater than the α-sequence of any connected component of M. Consequently, β_M is strictly less than β_J; and M satisfies π by the minimality of β_J. □

Lemma 3.10 *Any graph N consisting of disjoint copies of $J_2, J_3, \ldots, J_k$ plus any number of disjoint copies of the graph M from Lemma* 3.9 *satisfies π.*

Proof By the proof of Lemma 3.9, the graph M is such that the maximal component of β_M is strictly less than α_{J_1}. By reasoning as we did in the proof of Lemma 3.8, it follows that β_N is strictly less than β_J and so N satisfies π by the minimality of β_J. □

Throughout, we refer to a G-vertex in G' and the corresponding vertex in G by the same name (and also to a vertex of P and the corresponding vertex in the portion of the partial ordering P' corresponding to P by the same name).

Consider the algorithm GREEDY(π) on input (G', P', s', t'). The partial ordering P' consists of a linear ordering, whose vertices are S_0, say, concatenated with the partial ordering P. The subgraph of G' induced by the vertices of S_0 is as the graph K of Lemma 3.8 and consequently every vertex of S_0 is always placed in every output from GREEDY(π). Note that the algorithm GREEDY(π) on input (G', P', s', t') with current-vertex s is working with exactly the same partial ordering, namely P, as is the algorithm

GREEDY(independent set) on input (G, P, s, t) with current-vertex s.

Suppose, as our induction hypothesis, that:

- the algorithm GREEDY(independent set) on input (G, P, s, t) has current-vertex u, for some descendant u of s in P, and has so far output the set of vertices S;
- the algorithm GREEDY(π) on input (G', P', s', t') has current vertex u in P' and has so far output the set of vertices $S_0 \cup S$; and
- the subgraph of G' induced by the vertices of $S_0 \cup S$ is in the form of a subgraph of the graph N in Lemma 3.10.

Note that the induction hypothesis clearly holds, in the base case, when the vertex u is actually s.

Suppose that the algorithm GREEDY(π) outputs the vertex u. If u is such that adding u to $S_0 \cup S$ completes a copy of $I_0 \cup \{c\}$ then we would have a copy of J within the subgraph of G' induced by the vertices of $S_0 \cup S \cup \{u\}$. This would yield a contradiction because this subgraph satisfies π (by definition), π is hereditary on induced subgraphs, and J would then have to satisfy π. Hence, the vertex u is not joined to any vertex of S in G and so u is output by the algorithm GREEDY(independent set).

Conversely, if the algorithm GREEDY(independent set) outputs u then this is because $S \cup \{u\}$ is an independent set in G; and consequently $S_0 \cup S \cup \{u\}$ induces in G' a subgraph of the form of a subgraph of the graph N in Lemma 3.10. Hence, by Lemma 3.10, u is output by the algorithm GREEDY(π).

By induction, we obtain that if S is a set of vertices output by the algorithm GREEDY(independent set) on input (G, P, s, t) then $S_0 \cup S$ is output by the algorithm GREEDY(π) on input (G', P', s', t'), and conversely. Hence, we have a reduction from $\mathcal{H}$ to $\mathcal{G}$, and this reduction can clearly be completed using logspace. $\square$

As Miyano did in [29], we can now remove the reliance in Theorem 3.7 that π is satisfied by all sets of independent edges. In order to do so, we will use Ramsey's Theorem [35].

Theorem 3.11 (Ramsey's Theorem) *Let t, q and r be given positive integers such that $q \geq r$. Then there exists a number m, whose value depends on t, r, q, with the property that if the r-subsets (subsets of r elements) of any set S of $n \geq m$ elements are partitioned into t disjoint components $A_1, A_2, \ldots, A_t$ then there is a q-subset of S all of whose r-subsets belong to A_i, for some i.*

Let V be the set of vertices of a graph G, let $t = 2$ and let $r = 2$. Define A_1 to be the set of 2-subsets of V corresponding to the edges of G; and let A_2 be all other 2-subsets of V. That is, A_1 corresponds to the set of edges of G and A_2 to the set of non-edges. Applying Theorem 3.11 yields the following [1].

Corollary 3.12 *Given any positive integer q, there is a positive integer m, depending on q, such that every graph with at least m vertices contains either a clique or an independent set of size q.*

We can now prove the following lemma, also used in the proof of Theorem 4 in [27].

Lemma 3.13 *Any graph property π that is non-trivial on a class of graphs $\mathcal{C}$ and hereditary is either satisfied by all independent sets of vertices or by all cliques.*

Proof Let q be any positive integer. By Corollary 3.12, there exists a positive integer m such that any graph with at least m vertices contains either a clique or an independent set of size q. As property π is non-trivial on the class of graphs $\mathcal{C}$, there must be a graph G in $\mathcal{C}$ with at least m vertices satisfying π. As π is hereditary, it is satisfied by either every clique of size at most q or by every independent set of size at most q. This holds for any q, therefore the result follows. $\square$

We can now prove the following two corollaries.

Corollary 3.14 *Let π be a polynomial time testable, hereditary graph property non-trivial on planar bipartite graphs. The problem GREEDY(partial ordering, planar bipartite, π) is complete for* **NP**.

Proof For every n, there exists a number $b(n)$ (take for example $b(n) = 2n$) such that all planar bipartite graphs with $b(n)$ or more nodes contain an independent set of n nodes. As π is non-trivial on planar bipartite graphs, and hereditary, it follows that all independent sets of nodes satisfy π. The corollary now follows by Theorem 3.7. $\square$

Corollary 3.15 *Let π be a polynomial time testable, hereditary graph property non-trivial on undirected graphs. The problem GREEDY(partial ordering, undirected graphs, π) is complete for* **NP**.

Proof If property π is satisfied by all independent sets of vertices then the result follows using the techniques in the proof of Theorem 3.7. If this is not the case then for any hereditary property π, non-trivial on undirected graphs, we define the complementary property $\overline{\pi}$ as follows: a graph G satisfies $\overline{\pi}$ if, and only if, its complement $\overline{G}$ satisfies π. Clearly, the property $\overline{\pi}$ is hereditary and non-trivial on undirected graphs, as the class of undirected graphs is closed under complementation. As π is not satisfied by all independent sets then, by Lemma 3.13, π is satisfied by all cliques; and therefore $\overline{\pi}$ is satisfied by all independent sets, because the complement of a clique is an independent set. As the problems GREEDY(partial ordering, undirected graphs, π) and GREEDY(partial ordering, undirected graphs, $\overline{\pi}$) are logspace-equivalent, it is possible to prove our result by using the same techniques used in the proof of Theorem 3.7 (*i.e.*, prove the theorem for GREEDY(partial ordering, undirected graphs, $\overline{\pi}$) and then reduce to GREEDY(partial ordering, undirected graphs, π) by the map $(G, P, s, v) \to (\overline{G}, P, s, v)$). □

3.5 Directed graphs

Having considered properties on undirected graphs, we will now examine the problem GREEDY(partial ordering, $\mathcal{C}$, π) when we take $\mathcal{C}$ to be the class of directed graphs. As we shall see, this problem remains **NP**-complete. However, before stating the theorem, we will give some graph theoretic definitions that will be used during the proof.

A *complete symmetric* (CS) digraph is a directed graph $D = (V, E)$ such that for any pair of vertices v_1, v_2, where $v_1 \neq v_2$, $(v_1, v_2) \in E$ and $(v_2, v_1) \in E$. A *complete antisymmetric transitive* (CAT) digraph is a digraph $D = (V, E)$, where $V = \{1, \ldots, n\}$ and is such that for all $1 \leq i < j \leq n$, $(i, j) \in E$ but $(j, i) \notin E$. A digraph G is *connected* if the underlying undirected graph is. A *cut-point* of a connected digraph G is a vertex c such that its removal (along with its incident edges) from G results in a digraph with at least 2 connected components. If a connected digraph does not have any cut-points then it is *biconnected.* These definitions will apply to our instance digraph G: our partial ordering P remains an acyclic directed graph.

Theorem 3.16 *Let π be a polynomial time testable, hereditary, non-trivial property on directed graphs. The problem GREEDY(partial ordering, directed graphs, π) is complete for* **NP**.

Proof Ramsey's Theorem is widely applicable. If V is the set of vertices of a directed graph $D = (V, E)$, and we assume that the vertices are labelled $u_1, u_2, \ldots, u_n$, where n is the size of V, we can partition the 2-subsets of V in 4 components A_1, A_2, A_3 and A_4 as follows:

$$A_1 = \{\{u_i, u_j\} : (u_i, u_j), (u_j, u_i) \notin E\}$$
$$A_2 = \{\{u_i, u_j\} : (u_i, u_j), (u_j, u_i) \in E\}$$
$$A_3 = \{\{u_i, u_j\} : (u_i, u_j) \in E, (u_j, u_i) \notin E, i < j\}$$
$$A_4 = \{\{u_i, u_j\} : (u_i, u_j) \notin E, (u_j, u_i) \in E, i < j\}$$

Any subgraph of D induced by a subset S_1 of V, such that all the 2-subsets of S_1 are in A_1 is an independent set. Any subgraph of D induced by a subset S_2 of V, such that all the 2-subsets of S_2 are in A_2, is a CS, as there is an edge between any two vertices in S_2. It is also not difficult to see that any subgraph of D induced by a subset S_3 of V, such that all the 2-subsets of S_3 are in A_3, is a CAT digraph; and so is any subgraph of D induced by a subset of V such that all the 2-subsets of the set is in A_4 (after an appropriate relabelling of the vertices). It follows by Ramsey's Theorem that for any positive integer $q \geq 2$, there is a positive integer m, whose value depends on q, such that every directed graph with at least m vertices contains either a) an independent set, or b) a complete symmetric digraph, or c) a complete antisymmetric transitive digraph of q vertices.

Since property π is hereditary and non-trivial on the class of directed graphs, it follows that π must be satisfied by a) all independent sets, b) by all CS digraphs or c) by all CAT digraphs. We will now show that the theorem can be proved in all 3 cases.

Case a) Property π is satisfied by all independent sets of vertices, therefore the technique used in the case of undirected graphs, *i.e.*, the technique used in the proof of Theorem 3.7, can be used here as well. We reduce from the problem GREEDY(partial ordering, undirected graphs, independent set). The definition of α- and β-sequences of directed graphs follows immediately from the definition of α- and β-sequences relative to undirected graphs. The only difference in the proof derives from the fact that the forbidden graph for property π is now a directed graph, and therefore applying the construction explained in the theorem results in a digraph.

Case b) For any hereditary property π, non-trivial on directed graphs, we will consider the complementary property $\overline{\pi}$. Clearly, the property $\overline{\pi}$ is non-trivial on directed graphs, as the class of directed graphs is closed under complementation and hereditary. If π is satisfied by all complete symmetric graphs then $\overline{\pi}$ is satisfied by all independent sets, because the complement of a CS digraph is an independent set. As the problems GREEDY(partial ordering, directed graphs, π) and GREEDY(partial ordering, directed graphs, $\overline{\pi}$) are logspace-equivalent, it is possible to prove the theorem by using the same technique used in Case a).

Case c) Using the same strategy seen in Theorem 7 of [29], we will assume that π is satisfied by all CAT digraphs but not by all independent sets. If π were satisfied by all independent sets then we could prove the theorem as in Case a). Let s be the largest integer such that any graph consisting of s independent vertices $u_1, u_2, \ldots, u_s$ and a CAT digraph of any size satisfies π, but there exists a CAT digraph C such that the directed graph consisting of C and $s+1$ independent vertices violates π. The existence of s derives from the fact that π is not satisfied by all independent sets.

To prove the theorem, we will reduce from GREEDY(partial ordering, undirected graphs, independent set). From an instance (G, P, 1, n) of this problem, where $G = (V, E), P = (V, D)$ and $V = \{1, 2, \ldots, n\}$, we derive an instance (G', P', u_1, $v_{k,n}$) of GREEDY(partial ordering, directed graphs, π). Let k be the number of vertices in C. Graph $G' = (V', E')$ is a directed graph, and its vertex set is composed of s independent nodes, $u_1, u_2, \ldots, u_s$, plus k copies of V. The ith copy of V is denoted V_i, and its vertices are labelled $v_{i,1}, v_{i,2}, \ldots, v_{i,n}$.

The edge set E' of G' is defined as follows:

$$E' = \{(v_{p,j}, v_{q,j}) : 1 \leq p < q \leq k, 1 \leq j \leq n\}$$

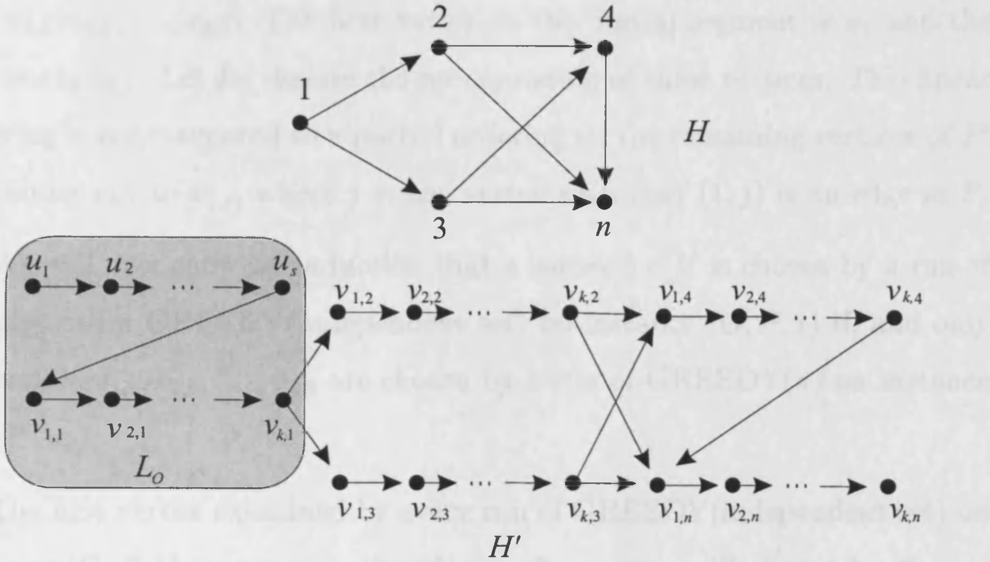


Figure 3.5: The partial ordering.

$$\cup \bigcup_{p=1}^{k} \{(v_{p,i}, v_{p,j}) : \{i,j\} \notin E, 1 \leq i < j \leq n\}$$

$$\cup \{(v_{p,i}, v_{q,j}) : 1 \leq p < q \leq k, 1 \leq i, j \leq n, \{i,j\} \notin E\}.$$

The construction of G' is such that for every i, where $1 \leq i \leq n$, vertices $v_{1,i}, v_{2,i}, \ldots, v_{k,i}$ form a CAT of size k.

The ordering on the vertices of G' is given by $P' = (V', D')$, where

$$D' = \{(u_i, u_{i+1}) : 1 \leq i \leq s-1\} \cup \{(u_s, v_{1,1})\}$$

$$\cup \{(v_{i,j}, v_{(i+1),j} : 1 \leq i \leq k-1, 1 \leq j \leq n\} \cup \{(v_{k,i}, v_{1,j}) : (i,j) \in D\}$$

See Figure 3.5 for an example.

The ordering P' begins with a linear ordering on the vertices $u_1, u_2, \ldots, u_s$

and $v_{1,1}, v_{2,1}, \ldots, v_{k,1}$. The first vertex in this initial segment is u_1 and the last one is $v_{k,1}$. Let L_O denote the set consisting of these vertices. This linear ordering is concatenated to a partial ordering on the remaining vertices of P' by joining $v_{k,1}$ to $v_{1,j}$, where j is any vertex such that $(1, j)$ is an edge in P.

We will now show by induction that a vertex $b \in V$ is chosen by a run of the algorithm GREEDY(independent set) on instance $(G, P, 1)$ if, and only if, vertices $v_{1,b}, v_{2,b}, \ldots, v_{k,b}$ are chosen by a run of GREEDY(π) on instance (G', P', u_1).

The first vertex examined by every run of GREEDY(independent set) on instance $(G, P, 1)$ is vertex 1, therefore such a vertex will always be chosen. By construction the subgraph of G' induced by $v_{1,1}, v_{2,1}, \ldots, v_{k,1}$ consists of a CAT of k vertices. It follows that the subgraph induced by the vertices of L_O consists of s independent vertices and a CAT of size k, and therefore it satisfies π. As every run of GREEDY(π) on instance (G', P', u_1) starts by examining the vertices in L_O, vertices $v_{1,1}, v_{2,1}, \ldots, v_{k,1}$ will always be chosen. This proves the base case of the following induction hypothesis.

Suppose, as our induction hypothesis, that:

- the algorithm GREEDY(independent set) on input $(G, P, 1)$ has so far chosen the set of vertices I, and vertex b is the next vertex to be examined;
- the algorithm GREEDY(π) on instance (G', P', u_1) has so far output the set of vertices $\{v_{1,l}, v_{2,l}, \ldots, v_{k,l} : l \in I\} \cup \{u_1, u_2, \ldots, u_s\}$, and the next vertex to be examined is $v_{1,b}$. As the vertices in I form an independent set of G, the subgraph of G' induced by $\{v_{1,l}, v_{2,l}, \ldots, v_{k,l} :$

$l \in I\}$ forms a CAT digraph; therefore the subgraph of G' induced by $\{v_{1,l}, v_{2,l}, \ldots, v_{k,l} : l \in I\} \cup \{u_1, u_2, \ldots u_s\}$ satisfies π.

When GREEDY(independent set) examines vertex b, if for some vertex $l \in I$, $(l, b) \in E$, then the subgraph induced by $I \cup \{b\}$ is not an independent set, and vertex b is not chosen.

The partial ordering on the vertices of G' is such that, for any two given vertices i and j in P, $(i, j) \in D$ if, and only if, in P' there is a path from $v_{1,i}$ to $v_{k,j}$ that visits vertices $v_{2,i}, v_{3,i}, \ldots, v_{k,i}$ and $v_{1,j}, v_{2,j}, \ldots, v_{k-1,j}$ (in this order).

For some vertex $l \in I, (l, b) \in E$; therefore in G', by construction, there will not be an edge between $v_{p,l}$ and $v_{p,b}$, for $1 \leq p \leq k$. Choosing any vertex from $\{v_{1,b}, v_{2,b}, \ldots, v_{k,b}\}$ would induce a subgraph of G' that contains $s + 1$ independent vertices and a CAT of size k, and therefore violates π. It follows that none of the vertices $\{v_{1,b}, v_{2,b}, \ldots, v_{k,b}\}$ will be chosen by GREEDY(π).

If vertex b is chosen by GREEDY(independent set) then it follows that for every vertex $l \in I$, $(l, b) \notin E$. The subgraph of G' induced by the set of vertices $\{v_{1,l}, v_{2,l}, \ldots, v_{k,l} : l \in I\} \cup \{v_{1,b}, v_{2,b}, \ldots, v_{k,b}\}$ forms a CAT digraph; therefore vertices $v_{1,b}, v_{2,b}, \ldots, v_{k,b}$ will be chosen by GREEDY(π) (as the other vertices chosen so far form an independent set of size s).

Instance $(G, P, 1, n)$ is a yes-instance of GREEDY(partial ordering, undirected graphs, independent set) if, and only if, vertex n appears in one of the subgraphs returned by GREEDY(independent set) on instance $(G, P, 1)$. This is the case if, and only if, for some run of GREEDY(π) on instance (G', P', u_1), vertices $\{v_{1,n}, v_{2,n}, \ldots, v_{k,n}\}$ appear in a subgraph of G' satisfying

π. Instance $(G', P', u_1, v_{k,n})$ is a yes-instance of GREEDY(partial ordering, directed graphs, π) if, and only if, vertex $v_{k,n}$ appears in a solution returned by the algorithm. It follows that $(G, P, 1, n)$ is a yes-instance if, and only if, $(G', P', u_1, v_{k,n})$ is a yes-instance. As the construction of $(G', P', u_1, v_{k,n})$ from $(G, P, 1, n)$ can be completed using logspace, the result follows. $\square$

3.6 Conclusion

In this chapter we proved several general results concerning the complexity of the problem GREEDY(partial ordering, $\mathcal{C}$, π) when π is a property hereditary, non-trivial on $\mathcal{C}$ and testable in deterministic polynomial time. When the class of graphs $\mathcal{C}$ under consideration is the class of undirected graphs, we proved in Corollary 3.14 that we can impose additional restrictions to our instance graphs so that the problem remains **NP**-complete. In particular, we proved that the problem GREEDY(partial ordering, planar bipartite, π) is complete for **NP**. It is of interest to consider what restrictions we can impose on the maximum degree of the vertices of a graph G in an instance of GREEDY(partial ordering, undirected graphs, π) without changing the complexity of the problem. Of course these restrictions will vary according to the property π, and we cannot therefore state a general result; but we can examine specific properties and try to obtain the boundary between **P** and **NP** for each one of them. We will focus on such problems in the next chapter.

Chapter 4

Boundaries between P and NP

4.1 Introduction

In this chapter we will examine the problem GREEDY(partial ordering, $\mathcal{C}$, π) for some specific properties π which are hereditary, testable in polynomial time and non-trivial on the class of undirected graphs. For every one of these properties we know from Corollary 3.15 that the problem is **NP**-complete. We will now establish some restrictions that we need to impose on the class of graphs $\mathcal{C}$ so that the problem becomes solvable in polynomial time. Like Miyano did in [29], we will consider restrictions on the maximum degree of the vertices of our instance graph.

For each of these properties we will show that by applying certain restrictions the problem still remains **NP**-complete, and we will show how the problem can be solved in deterministic polynomial time if the restrictions imposed become more severe.

In order to show that the problem GREEDY(partial ordering, $\mathcal{C}$, π) is **NP**-complete for a specific $\mathcal{C}$ and a particular π, we will always reduce from the problem 3-SAT. As all reductions follow the same pattern, we will describe the reduction scheme for an unspecified property π that satisfies the aforementioned requirements and then, for each considered property, we will only show the part of the reduction that is particular to π. We will therefore use a strategy similar to the one used by Miyano, with the difference that he reduces from different versions, depending on the property π, of the circuit value problem and only deals with linear orders (for more details see [29]). The reader might wonder why we did not reduce from the problem GREEDY(partial ordering, $\mathcal{C}$, π) itself, instead of taking the problem 3-SAT as our base case. The reason is that by using such a problem we have managed to devise a reduction scheme that is applicable to all the properties considered in this chapter, and which might be used in the future for other properties not considered here.

4.2 The reduction scheme

We will reduce from the known **NP**-complete problem 3-SAT, whose instances are: a set of literals $X = \{x_1, \neg x_1, x_2, \neg x_2, \ldots, x_n, \neg x_n\}$ and a sequence of clauses $C = (c_1, c_2, \ldots, c_m)$ where each clause c_i is a subset of X containing 3 elements. Yes-instances of 3-SAT are instances such that there is a truth assignment on the Boolean variables $x_1, x_2, \ldots, x_n$ that satisfies all the clauses in C, *i.e.*, a subset $X' \subseteq X$ such that $|X' \cap \{x_i, \neg x_i\}| = 1$, for $1 \leq i \leq n$, and such that $|X' \cap c_j| \geq 1$, for $1 \leq j \leq m$.

We refer to the literals appearing in clause c_i as $l_{i,1}, l_{i,2}, l_{i,3}$ and to their negations as $\overline{l_{i,1}}, \overline{l_{i,2}}, \overline{l_{i,3}}$. So, for example, if clause c_3 is $(\neg x_1 \vee x_2 \vee x_3)$ then $l_{3,1} = \neg x_1$, $l_{3,2} = x_2$ and $l_{3,3} = x_3$ while $\overline{l_{3,1}} = x_1$, $\overline{l_{3,2}} = \neg x_2$ and $\overline{l_{3,3}} = \neg x_3$ (note that it might be the case that $l_{3,1} = \overline{l_{3,3}}$, for example).

From an instance (C, X) of 3-SAT we shall derive an instance (G, P, s, t) of GREEDY(partial ordering, $\mathcal{C}$, π) where: $\mathcal{C}$ is our chosen class of graphs and G is an undirected graph belonging to $\mathcal{C}$; P is a partial ordering on the vertices of G; and s and t are two distinguished vertices. What is more, (C, X) will be a yes-instance if, and only if, (G, P, s, t) is a yes-instance. The construction will be such that it can be completed using logspace. All instance graphs G obtained in the reductions will have the same underlying structure. They are built using blocks, whose form depends on the considered property π, joined according to a template that we will define below. Essentially, the template consists of a skeleton containing 'empty spaces' that will be filled by inserting gadgets particular to each property π. There are two types of gadgets and we will call them, respectively, O and A: these gadgets will be given for each property π in the corresponding section.

Gadget O contains, regardless of the property π, 7 vertices which are labelled, respectively, o, l_1, l_2, l_3 and $\overline{l_1}, \overline{l_2}, \overline{l_3}$. It also contains a number of vertices labelled $b_1, b_2, \ldots, b_k$, where k is a number that depends on π. We will refer to such a set of vertices as *b-vertices.* The gadget A contains a vertex labelled a and two vertices labelled, respectively, u_1 and u_2. As in the case of O, such vertices do not depend on π. The gadget A also contains a number of vertices labelled, respectively, $e_1, e_2, \ldots, e_r$, where r is determined by π. Such vertices will be called *e-vertices.* The gadgets are used to emulate the structure of a Boolean formula: O represents a clause and A represents the conjunction of two clauses.

The graph G is constructed from C as follows. For each clause c_i we add to G a copy of gadget O, and we call such a copy O_i (all copies are disjoint). It follows that the graph will contain m copies of O, where m is the number of clauses in C. We will label the vertices of O_i in the following way. Vertex o is labelled o_i. Vertices l_1, l_2, l_3 are labelled as $l_{i,1}, l_{i,2}, l_{i,3}$ respectively, where $l_{i,1}, l_{i,2}$ and $l_{i,3}$ are the literals appearing in clause i. We will call vertices of the form $l_{\_,\_}$ *literal-vertices.* We will label vertices $\overline{l_1}, \overline{l_2}, \overline{l_3}$ as $\overline{l_{i,1}}, \overline{l_{i,2}}, \overline{l_{i,3}}$ respectively; that is, with the negations of the literals appearing in clause i. We will refer to vertices of the form $\overline{l_{\_,\_}}$ as *negated-vertices.* Note that there might be more than one vertex in G with the same label. All b-vertices in O, that are of the form b_j, will now be labelled as $b_{i,j}$, depending on which copy of O_i they lie in.

The graph G also contains $(m-1)$ disjoint copies of gadget A: these copies will be referred to as $A_1, A_2, \ldots, A_{m-1}$. In each copy A_i we will label vertex a as a_i. We will also label the e-vertices in each copy of A as we did in the case of the b-vertices; that is, every node labelled e_j will be labelled $e_{i,j}$, depending upon which copy of A_i it lies in.

To construct our template we then proceed as follows. In copy A_1 we will identify vertex u_1 with o_1 from O_1 and u_2 with vertex o_2 from O_2. In every other gadget A_i, where $2 \leq i \leq m-1$, we will identify u_1 with vertex a_{i-1} from A_{i-1} and identify vertex u_2 with o_{i+1} from O_{i+1}.

The construction is concluded by adding an independent vertex named

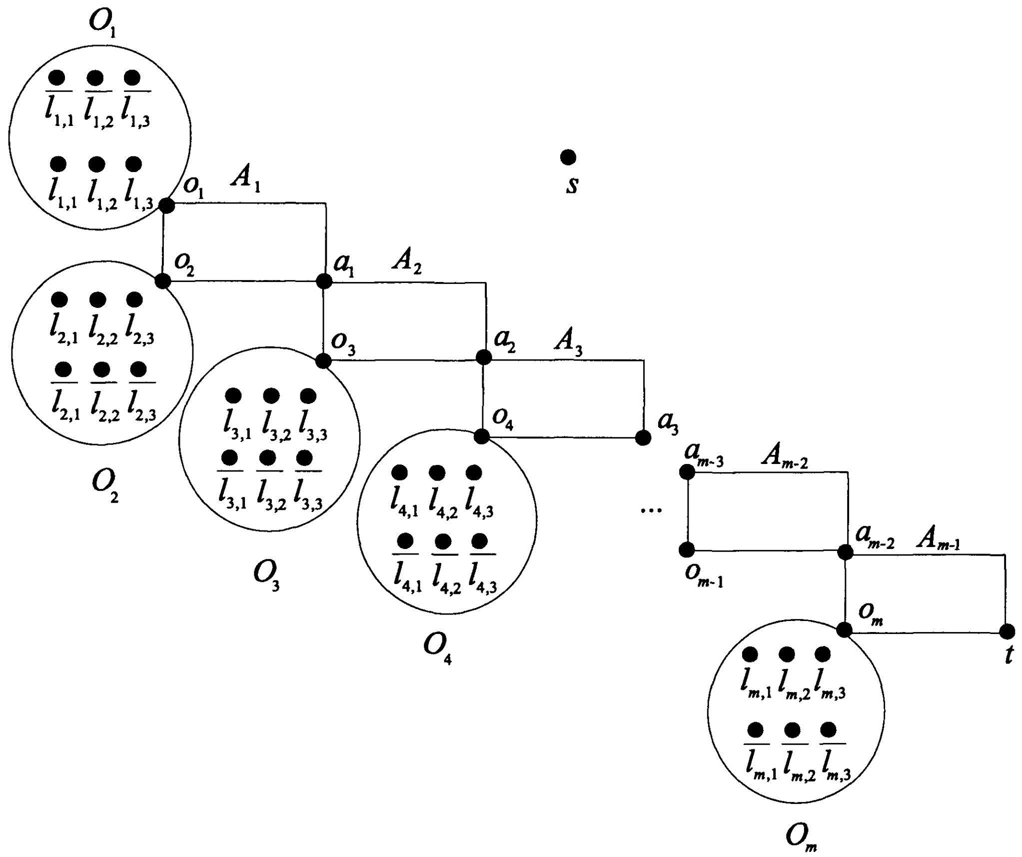


Figure 4.1: The structure of G.

(and corresponding to) s, and by renaming the vertex a_{m-1} as t. In Figure 4.1 we show an illustration of the structure of G. In the figure we represent each copy of graph O_i as an oval, and each copy of A_i as a rectangle (note that e-vertices and b-vertices are not shown to avoid cluttering the figure). To obtain the finished graph it is only necessary to insert in place of the ovals and rectangles the gadgets O and A that are relevant to each considered property.

The partial ordering P is defined as follows. We will start by ordering the Boolean variables $\{x_1, x_2, \ldots, x_n\}$ lexicographically in the obvious way, that is:

$$x_1 < x_2 < \ldots < x_n$$

and denote such an ordering as $<_x$. We then divide literal-vertices and negated-vertices according to the associated literals. All vertices labelled with literals of the form x_- will be called *positive*-vertices, while all labelled with an associated literal of the form $\neg x_-$ will be called *negative*-vertices. We can now order all positive-vertices according to $<_x$. So if a vertex g_1 is less than a vertex g_2 in this ordering then the associated literals are such that the one associated with g_1 is less than or equal to the literal associated with g_2. We proceed analogously to order the negative-vertices by taking complements. By construction of G, for every positive-vertex there is a corresponding negative-vertex, and vice versa, and the two corresponding vertices are in the same position in both orderings. By this we mean that if a positive-vertex is, say, the third in the ordering of the positive-vertices then the negative-vertex corresponding to it will be the third in the ordering of the negative-vertices. We will denote the ordered positive-vertices as $\lambda_1, \lambda_2, \ldots, \lambda_k$, and the ordered negative-vertices as $\mu_1, \mu_2, \ldots, \mu_k$.

The partial ordering P begins as follows. The vertex s is less than λ_1 and μ_1. We then have the orderings $\lambda_1 < \lambda_2 < \ldots < \lambda_k$ and $\mu_1 < \mu_2 < \ldots < \mu_k$. For any index $i \in \{1, 2, \ldots, k-1\}$, if the associated literal of λ_i is different from the literal associated with λ_{i+1} then we also have $\lambda_i < \mu_{i+1}$ and $\mu_i < \lambda_{i+1}$. The partial ordering continues with a linear ordering on the vertices

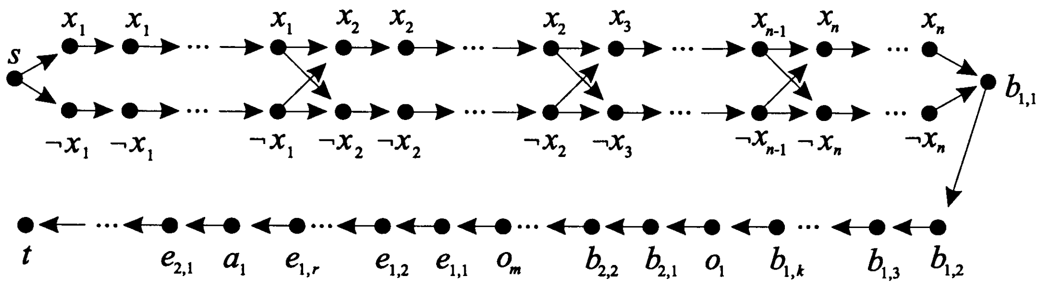


Figure 4.2: The partial ordering.

of each copy of A and O which are not literal- or negated-vertices. Such an ordering is defined as follows:

$$b_{1,1} < b_{1,2} < \ldots < b_{1,k} < o_1 < b_{2,1} < b_{2,2} < \ldots < o_2 < \ldots < o_m$$

$$o_m < e_{1,1} < e_{1,2} < \ldots < e_{1,r} < a_1 < e_{2,1} < e_{2,2} < \ldots < a_2 < \ldots < t$$

Finally we define that both λ_k and μ_k are less that $b_{1,1}$. Note that we chose t to be vertex a_{m-1}.

The partial ordering P emulates an assignment of truth values to the variables $x_1, x_2, \ldots, x_n$. Each path starts from vertex s and visits either all positive-vertices labelled x_1 or all negative-vertices labelled $\neg x_1$, then it visits either all positive-vertices labelled x_2 or all negative-vertices labelled $\neg x_2$, and so on until vertex $b_{1,1}$ is reached. It is important to note that for any pair of vertices $\{x_i, \neg x_i\}$, exactly one of the components will appear on the path, thus ensuring that the corresponding truth assignment is valid. The remaining vertices of P induce a linear ordering on the other vertices of G. An example of the structure of P is shown in Figure 4.2.

We will now explain the mechanism of the reduction. By construction every execution of GREEDY(π) on instance (G, P, s) starts by choosing a path in the partial ordering on the vertices of G corresponding to a truth assignment on the variables in X. A variable x_i in X is assigned the value true if a vertex labelled x_i appears on the path, and it is assigned the value false if a vertex labelled $\neg x_i$ appears on the path.

For each property π, the graph G obtained after inserting in our template the copies of the appropriate gadgets A and O has the characteristic (and this will be clearly visible when we will show the gadgets in the relevant sections) that the subgraph of G induced by the literal-vertices and by the negated-vertices is an independent set. Note that all properties π with which we will use this reduction scheme are satisfied by all independent sets of vertices. As every execution of GREEDY(π) on instance (G, P, s) starts by visiting s and a collection of literal and negated-vertices, and π is satisfied by all independent sets of vertices, it follows that all such vertices will be selected. By construction, in P every path continues by visiting all b-vertices in O_1, and then vertex o_1. For all reductions, the gadget O is such that the set of vertices chosen by GREEDY(π) on instance (G, P, s) will contain vertex o_1 if, and only if, at least one of $\{l_{1,1}, l_{1,2}, l_{1,3}\}$ has been chosen. If this was not the case then choosing vertex o_1 would induce a subgraph of G that violates property π. This is achieved by using gadgets that consist of forbidden graphs for the property "glued" together. For example, in the case of property 3-cycle free the gadgets consist of collections of triangles (see Figure 4.3 on page 62). So, if vertex o_1 is chosen then at least one of the literals appearing in clause c_1 in C evaluates to true in the assignment corresponding to the chosen path. The same process is repeated for all vertices in $O_2, O_3, \ldots, O_m$.

That is, vertex o_i is chosen if, and only if, at least one of $\{l_{i,1}, l_{i,2}, l_{i,3}\}$ appears on the path.

When all vertices in O_m have been examined the algorithm proceeds by examining the vertices in $A_1 \setminus \{o_1, o_2\}$. If both o_1 and o_2 have previously been chosen then vertex a_1 will be selected, else it will be rejected. This is because of the construction of the gadget A. If one (or both) of the vertices o_1 and o_2 were previously rejected then the algorithm would proceed by choosing a set of vertices which, with vertex a_1, form a subgraph of G that violates property π. It follows that vertex a_1 will be rejected. The execution then proceeds by visiting the vertices in $A_2 \setminus \{a_1, o_3\}$. If both a_1 and o_3 have been chosen then vertex a_2 will be selected, else a_2 will be rejected. So vertex a_2 is chosen if, and only if, vertices o_1, o_2 and o_3 have been chosen.

The process is repeated until the algorithm examines the vertices of $A_{m-1} \setminus \{a_{m-2}, o_m\}$. If both vertices a_{m-2} and o_m have been selected then vertex a_{m-1}, which has been designated vertex t, will be chosen, else it will be rejected. This will only happen if vertices $o_1, o_2, \ldots, o_m$ have previously been selected. It is straightforward to notice that if vertex t is chosen then the truth assignment corresponding to the path chosen in P will satisfy (C, X). Conversely, if there exists a satisfying truth assignment for (C, X) then the corresponding path in P will induce an execution of GREEDY(π) on instance (G, P, s) that results in vertex t being chosen. This means that (G, P, s, t) is a yes-instance of GREEDY(π) if, and only if, there exists a truth assignment satisfying C. Note that for all the properties considered, the construction can be completed using logspace.

No.	Property	Restriction	Degree
(1)	3-cycle free	planar	4
(2)	k-cycle free $k \geq 5$	planar	3
(3)	bipartite	planar	3
(4)	planar	bipartite	3
(5)	outerplanar	planar and bipartite	3
(6)	edge graph	planar and bipartite	3
(7)	interval graph	planar and bipartite	3
(8)	acyclic	planar and bipartite	3
(9)	chordal	planar and bipartite	3
(10)	4-cycle free	planar and bipartite	4
(11)	maximum degree 1	planar and bipartite	3
(12)	independent set	planar and bipartite	3

Table 4.1: NP-complete problems

In the following section we will give the gadgets O and A for the considered properties, and will explain why vertices o_i and a_i in each copy of the gadgets are chosen or rejected according to the preceding explanation.

4.3 Optimal degree bounds

We will now give the optimal degree bound for some specific properties π. For each considered property we will exhibit the gadgets O and A and will illustrate the details of the proof which were previously omitted. Note that by construction of our linear ordering P, for every gadget O_i exactly three of the vertices in $\{l_{i,1}, l_{i,2}, l_{i,3}, \overline{l_{i,1}}, \overline{l_{i,2}}, \overline{l_{i,3}}\}$ appear on every path from vertex s to vertex t; and note also that such vertices will be examined before the corresponding b-vertices and o_i. Our results are summarised in Table 4.1, numbers 1-9. The column *Property* indicates the property π under consideration. The columns *Restriction* and *Degree*, together, characterise the class $\mathcal{C}$ of undirected graphs considered (the ordering is always a partial ordering). So, for example, the first row of the table indicates that the problem GREEDY(partial ordering, planar with maximum degree 4, 3-cycle free) is **NP**-complete. We will discuss the various problems by following the order of the table 4.1; that is, we will begin with the property 3-cycle free.

4.3.1 3-cycle free

Lemma 4.1 *The problem GREEDY(partial ordering, planar with maximum degree* 4, 3*-cycle free) is* **NP***-complete.*

Proof We explained in the reduction scheme that the graph G is constructed from C by connecting m copies of the gadget O and $m-1$ copies of gadget A. When our property π is 3-cycle free (a graph is k-cycle free if it does not contain a cycle of length k), the gadgets are as in Figure 4.3, where we show the first copies of the graphs, O_1 and A_1.

We will now describe an execution of the algorithm GREEDY(3-cycle free) on an instance (G, P, s). We will assume that some positive- and negative-vertices have been chosen and that the next vertex on the path is $b_{1,1}$. We will show that if at least one of $l_{1,1}, l_{1,2}, l_{1,3}$ has been chosen so

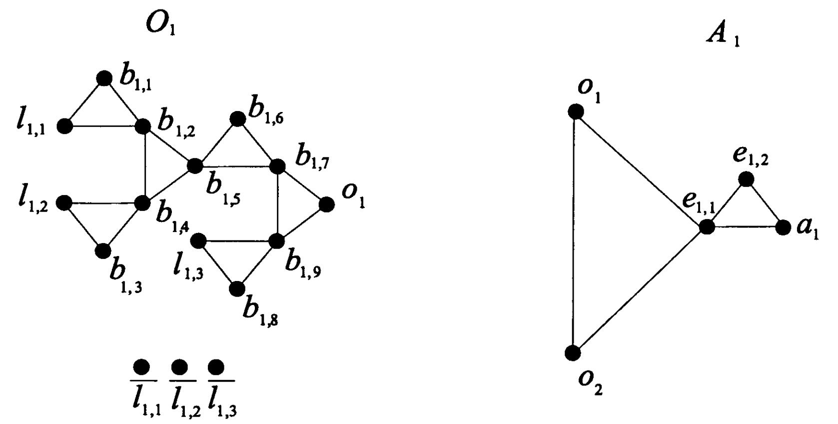


Figure 4.3: O_1 and A_1 for property 3-cycle free.

far by the algorithm then this will result in vertex o_1 being chosen. If all vertices in $\{\overline{l_{i,1}}, \overline{l_{i,2}}, \overline{l_{i,3}}\}$ are chosen then vertex o_1 will be rejected. Note that, by construction of our partial ordering P, $b_{1,1} < b_{1,2} < \ldots < b_{1,9} < o_1$.

In the O gadget O_1, if $l_{1,1}$ is chosen then $b_{1,1}$ will be chosen, but $b_{1,2}$ will be rejected. This means that vertex $b_{1,5}$ will be chosen, regardless of whether $b_{1,4}$ is selected or not. Vertex $b_{1,7}$ will then be rejected and vertex o_1 will therefore be chosen. If $l_{1,2}$ is chosen then vertex $b_{1,4}$ will be rejected and therefore $b_{1,5}$ will be chosen, resulting in o_1 being chosen. If $l_{1,1}$ and $l_{1,2}$ are not chosen, but vertex $l_{1,3}$ is, then this means that vertex $b_{1,7}$ will be chosen, but $b_{1,9}$ will be rejected, and therefore o_1 will be chosen.

Conversely, if none of $l_{1,1}, l_{1,2}, l_{1,3}$ is chosen then $\overline{l_{1,1}}, \overline{l_{1,2}}, \overline{l_{1,3}}$ are, and therefore both vertices $b_{1,2}$ and $b_{1,4}$ will be chosen; this means that $b_{1,5}$ will be rejected. It follows that $b_{1,7}$ will be chosen. As $l_{1,3}$ has not been chosen, vertex $b_{1,9}$ will be chosen, which implies that o_1 is rejected. Although the example is given on O_1, this clearly holds for all vertices o_i. That is, if at least one of $l_{i,1}, l_{i,2}, l_{i,3}$ is chosen then o_i will also be chosen, while if all of $\overline{l_{1,1}}, \overline{l_{1,2}}, \overline{l_{1,3}}$ are chosen then o_i will be rejected.

In the case of gadget A_1, if o_1 and o_2 are chosen then vertex $e_{1,1}$ will be rejected, which in turn implies that vertex a_1 will be chosen. If at most one of o_1 and o_2 is chosen then vertex $e_{1,1}$ will be chosen, which means that a_1 will be rejected. The execution continues by visiting all vertices in $A_2 \setminus \{o_3, a_1\}$. If both o_3 and a_1 were previously chosen then vertex a_2 will also be chosen, otherwise it will be rejected. This holds for all a_i, where $2 \leq i \leq m-1$. As vertex a_{m-1} is our vertex t, it will be chosen if, and only if, all vertices o_i, where $1 \leq i \leq m$, have been chosen. This, as was explained in the description of the template, means that the truth assignment corresponding to the chosen path in P satisfies all the clauses in C. Vice versa, similar reasoning yields that if a truth assignment satisfying all the clauses in C exists then the corresponding path will be chosen by an execution of GREEDY(3-cycle free) on instance (G, P, s); and the output of the execution will contain vertex t.

The only vertices that connect copies of O and A gadgets are of the form o_- and a_-; it is therefore clear that the graph G is planar and every vertex has degree at most 4. The result follows. □

We will now show that if we consider graphs with degree at most 3, the problem GREEDY(partial ordering, maximum degree 3, 3-cycle free) can be solved in polynomial time. Let us consider an instance (G, P, s, t) of the problem GREEDY(partial ordering, maximum degree 3, 3-cycle free). For any 3 vertices that induce a triangle in G, and that appear on a path in P, in a particular execution of GREEDY(3-cycle free) only the first 2 vertices appearing on the path will be chosen. Also, any 3 vertices that do not induce a triangle in G and appear on a path in P will all be chosen.

We will often need to check if a vertex is reachable from another in a directed graph P. It is well known that this problem is solvable in polynomial time [31]. We also say that vertex t is reachable in $P \setminus S$, where S is a set of vertices, when we mean that there is a path in P from s that reaches our chosen vertex without encountering any of the nodes in S on the path. If such a path exists then one of the executions of the algorithm GREEDY(3-cycle free) will choose it.

In any graph of degree at most 3, any vertex can be in at most 3 cycles of length 3. We therefore only need to examine a limited number of cases concerning vertex t and its neighbours. If $\deg_G(t) = 1$ the problem is trivial (t is always output). So we only need to focus on the 2 remaining possibilities, that is, $\deg_G(t) = 2$ and $\deg_G(t) = 3$. All cases refer to Figure 4.4. Case 1: $\deg_G(t) = 2$.

(a) As vertex t does not appear on any triangle, it will be chosen by any execution of the algorithm GREEDY(3-cycle free) such that t appears on the path.

(b) Vertex t will be chosen if, and only if, there exists at least one path from s to t in $P \setminus \{1\}$ or $P \setminus \{2\}$.

(c) If vertex t is reachable from s in $P \setminus \{1\}$ or $P \setminus \{2\}$ then it will be chosen. If vertex 1 or vertex 2 are reachable from vertex 3, for at least

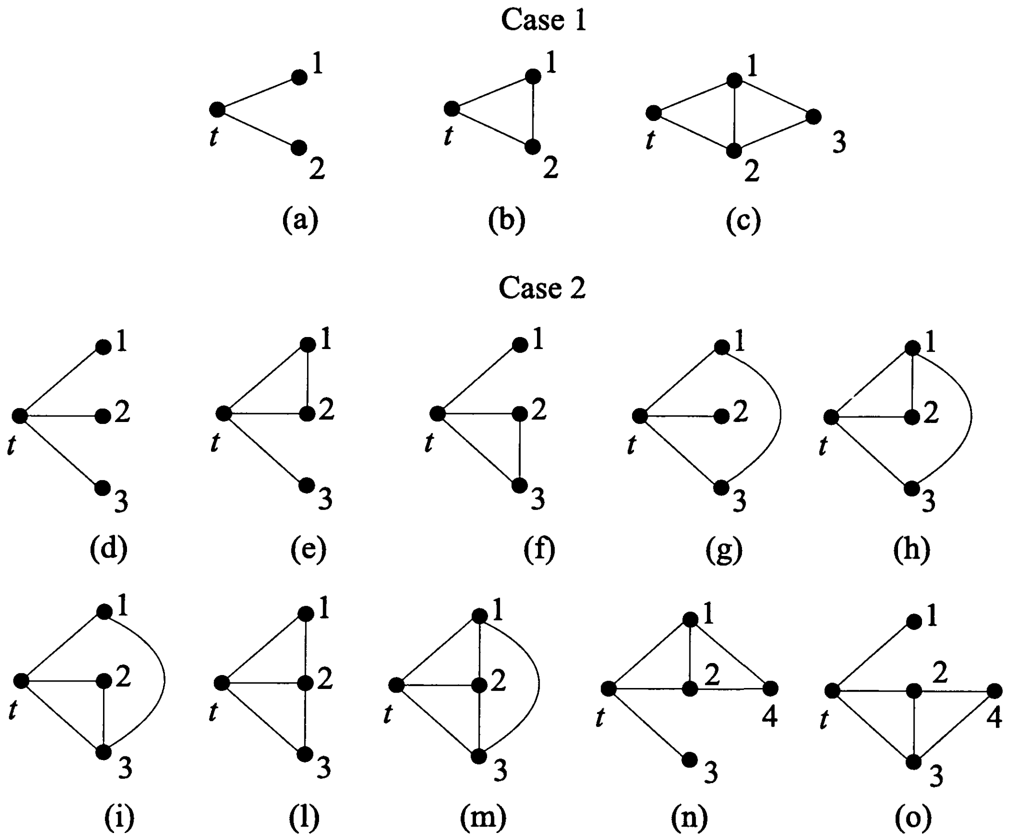


Figure 4.4: GREEDY(3-cycle free) for maximum degree 3

one execution of GREEDY(3-cycle free), vertex 3 will be chosen and at most one of 1, 2 will be selected. This means that t will be chosen as well. If none of these two conditions occurs, vertex t will not be chosen.

Case 2: $\deg_G(t) = 3$.

(d) Vertex t will be chosen by any execution of GREEDY(3-cycle free) such that t appears on the path.

(e) Vertex t will be chosen if, and only if, there exists at least one path from s to t in $P \setminus \{1\}$ or $P \setminus \{2\}$.

(f) Vertex t will be chosen if, and only if, there exists at least one path from s to t in $P \setminus \{2\}$ or $P \setminus \{3\}$.

(g) Vertex t will be chosen if, and only if, there exists at least one path from s to t in $P \setminus \{1\}$ or $P \setminus \{3\}$.

(h) Vertex t will be chosen if, and only if, there exists at least one path from s to t in $P \setminus \{1\}$ or $P \setminus \{2, 3\}$.

(i) Vertex t will be chosen if, and only if, there exists at least one path from s to t in $P \setminus \{3\}$ or $P \setminus \{1, 2\}$.

(l) Vertex t will be chosen if, and only if, there exists at least one path from s to t in $P \setminus \{2\}$ or $P \setminus \{1, 3\}$.

(m) Vertex t will be chosen if there exists at least one path from s to t in $P \setminus \{1, 2\}$ or $P \setminus \{1, 3\}$ or $P \setminus \{2, 3\}$.

(n) If vertex t is reachable from s in $P \setminus \{1\}$ or $P \setminus \{2\}$ then it will be chosen. If vertex 1 or vertex 2 are reachable from vertex 4, for at least one execution of GREEDY(3-cycle free), vertex 4 will be chosen and at most one of 1, 2 will be selected. This means that t will be chosen as well. If none of these two conditions occurs, vertex t will be rejected.

(o) If vertex t is reachable from s in $P \setminus \{2\}$ or $P \setminus \{3\}$ then it will be chosen. If vertex 2 or vertex 3 are reachable from vertex 4, for at least one execution of GREEDY(3-cycle free), vertex 4 will be chosen and at most one of 2, 3 will be selected. This means that t will be chosen as well. If none of these two conditions occurs, vertex t will be rejected.

It is clear that the structure of the connected component containing t, as shown in Figure 4.4, can be found in polynomial time, therefore the problem GREEDY(partial ordering, maximum degree 3, 3-cycle free) can also be solved in polynomial time.

4.3.2 k-cycle free, $k \geq 5$.

Lemma 4.2 *The problem GREEDY(partial ordering, planar with maximum degree* 3, *k-cycle free) is* **NP***-complete.*

Proof We describe the O and A gadgets with reference to Figure 4.5 where we take k to be 5. The size of the gadgets increases according to the value of k, but the structure remains the same. The only thing that changes is the length of the cycles. So, for example, if we take k to be 6 then the graph is composed of a collection of hexagons instead of a collection of pentagons. Note that the gadgets A_1 and A_i, for $2 \leq i \leq m-1$, are connected to the skeleton slightly differently. In A_1 e-vertex $e_{1,3}$ is joined to b-vertex $b_{1,13}$ from O_1, and e-vertex $e_{1,6}$ is joined to b-vertex $b_{2,13}$ from O_2. For all other A gadgets A_i, for $2 \leq i \leq m-1$, vertex $e_{i,3}$ is adjacent to vertex $e_{i-1,14}$ from A_{i-1}, and vertex $e_{i,6}$ is adjacent to b-vertex $b_{i+1,13}$ from O_{i+1}.

In O_1, if vertex $l_{1,1}$ is chosen then $b_{1,1}, b_{1,2}$ and $b_{1,3}$ are also chosen, while $b_{1,4}$ is rejected (if this was not the case then the induced subgraph would contain a cycle of length 5). This means that $b_{1,9}$ will be selected. Vertices

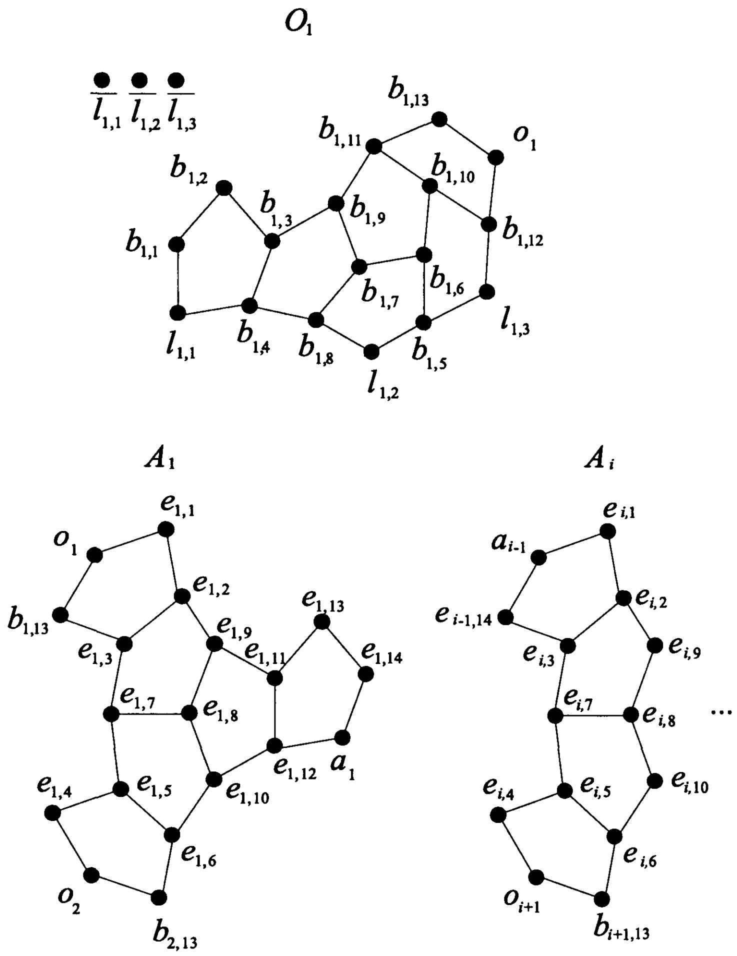


Figure 4.5: O_1, A_1 and A_i for property 5-cycle free.

$b_{1,6}$ and $b_{1,7}$ are always chosen, and $b_{1,10}$ will be selected. Vertex $b_{1,11}$ will be rejected and this results in vertex o_1 being chosen.

If vertex $l_{1,2}$ is chosen, $b_{1,5}, b_{1,6}$ and $b_{1,7}$ are chosen while vertex $b_{1,8}$ is rejected, and therefore $b_{1,9}$ is chosen. This, as explained before, results in o_1 being chosen. If $l_{1,3}$ is chosen then $b_{1,12}$ is rejected, and therefore o_1 will be chosen.

If none of $l_{1,1}, l_{1,2}, l_{1,3}$ are chosen then both $b_{1,4}$ and $b_{1,8}$ are selected, and therefore $b_{1,9}$ is rejected. This results in vertices $b_{1,10}$ and $b_{1,11}$ both being chosen. As $l_{1,3}$ was not selected then vertex $b_{1,12}$ is chosen. Vertex $b_{1,13}$ is selected and therefore o_1 is rejected.

When examining gadget A_1, vertices $b_{1,13}$ and $b_{2,13}$, that are part of O_1 and O_2, respectively, are always chosen. If vertices o_1 and o_2 are chosen then vertices $e_{1,3}$ and $e_{1,6}$ are rejected, and this means that vertices $e_{1,9}$ and $e_{1,10}$ will be chosen. Vertex $e_{1,12}$ will therefore be rejected and a_1 will therefore be chosen.

If o_1 is not chosen then $e_{1,3}$ is selected, and $e_{1,9}$ is therefore rejected. This results in vertex $e_{1,12}$ being chosen and in a_1 being rejected.

If vertex o_2 is not chosen then $e_{1,6}$ is chosen; this means that $e_{1,10}$ is rejected and $e_{1,12}$ is chosen, resulting in a_1 being rejected. It follows that a_1 will be chosen if, and only if, both o_1 and o_2 have been chosen. Note that vertex $e_{1,14}$ is always chosen, so the same holds for all gadgets A_i, for $2 \leq i \leq m-1$. Similar reasoning results in vertex t being chosen if, and only if, all vertices o_i, where $1 \leq i \leq m$, have been chosen. Therefore t is chosen if, and only if, the truth assignment corresponding to the chosen path in P satisfies all the clauses in C. Vice versa, similar reasoning yields that if a truth assignment satisfying all the clauses in C exists then the corresponding path will be chosen by an execution of GREEDY(k-cycle free) on instance (G, P, s); and the output of the execution will contain vertex t. It is not difficult to notice that the graph obtained by joining the O and A gadgets is planar and has maximum degree 3. $\square$

We will now show that the problem GREEDY(partial ordering, maximum degree 2, k-cycle free) is solvable in polynomial time. Note that any vertex can be in at most one cycle in a graph G of maximum degree 2. We will assume that vertex t is on a cycle of length k; if this was not the case then, as we assumed that every vertex is reachable from s in our partial order P, t would trivially be chosen by at least one execution of the algorithm. For every vertex $x \in G$ such that x is on the cycle containing t and $x \neq t$, we can check whether t is reachable from s in $P \setminus \{x\}$. If for any x this is true then for at least one execution of GREEDY(k-cycle free), vertex t will be chosen, and (G, P, s, t) will therefore be a yes-instance. If t is not reachable in $P \setminus \{x\}$, for any x on the cycle, then every path from s to t will visit all vertices on the cycle before reaching t, and all such vertices will be chosen. It follows that t will be rejected by every run of the algorithm, and so (G, P, s, t) is a no-instance of the problem.

4.3.3 Bipartite

Lemma 4.3 *The problem GREEDY(partial ordering, planar with maximum degree* 3, *bipartite) is* **NP***-complete.*

Proof It is well known that a bipartite graph does not contain any cycles of odd length. We will use this in our reduction to force all vertices o_i to be chosen if, and only if, at least one of the corresponding $l_{i,1}, l_{i,2}, l_{i,3}$ appears on a path, and is, by construction of G, chosen by an execution of

GREEDY(bipartite) on instance (G, P, s). We use the same strategy to force vertices a_i to be chosen or rejected appropriately.

The result is proved by using the same gadgets given for the property 5-cycle free: we simply take a 5-cycle as a forbidden subgraph for the property bipartite. With reference to Figure 4.5, we will now show that, by construction of our graph G, any set of vertices returned by an execution of GREEDY(bipartite) is such that the induced subgraph of G does not contain any cycles.

- In O_1, if none of $l_{1,1}, l_{1,2}$ and $l_{1,3}$ are chosen, the algorithm will choose vertices $b_{1,1}, b_{1,2}, b_{1,3}, b_{1,4}, b_{1,5}, b_{1,6}, b_{1,7}, b_{1,8}, b_{1,10}, b_{1,11}, b_{1,12}$ and $b_{1,13}$. The induced subgraph is clearly acyclic.
- If vertex $l_{1,1}$ is chosen, while $l_{1,2}$ and $l_{1,3}$ are not, the algorithm will choose vertices $b_{1,1}, b_{1,2}, b_{1,3}, b_{1,5}, b_{1,6}, b_{1,7}, b_{1,8}, b_{1,9}, b_{1,10}, b_{1,12}, b_{1,13}$ and o_1. The induced subgraph is acyclic.
- If vertex $l_{1,2}$ is chosen, while $l_{1,1}$ and $l_{1,3}$ are not, the algorithm will choose vertices $b_{1,1}, b_{1,2}, b_{1,3}, b_{1,4}, b_{1,5}, b_{1,6}, b_{1,7}, b_{1,9}, b_{1,10}, b_{1,12}, b_{1,13}$ and o_1. The induced subgraph is acyclic.
- If vertex $l_{1,3}$ is chosen, while $l_{1,1}$ and $l_{1,2}$ are not, the algorithm will choose vertices $b_{1,1}, b_{1,2}, b_{1,3}, b_{1,4}, b_{1,5}, b_{1,6}, b_{1,7}, b_{1,8}, b_{1,10}, b_{1,11}, b_{1,13}$ and o_1. The induced subgraph is acyclic.
- If vertices $l_{1,1}$ and $l_{1,2}$ are chosen, while $l_{1,3}$ is not, the algorithm will choose vertices $b_{1,1}, b_{1,2}, b_{1,3}, b_{1,5}, b_{1,6}, b_{1,7}, b_{1,9}, b_{1,10}, b_{1.12}, b_{1,13}$ and o_1. The induced subgraph is acyclic.

- If vertices $l_{1,1}$ and $l_{1,3}$ are chosen, while $l_{1,2}$ is not, the algorithm will choose vertices $b_{1,1}, b_{1,2}, b_{1,3}, b_{1,5}, b_{1,6}, b_{1,7}, b_{1,8}, b_{1,9}, b_{1,10}, b_{1,13}$ and o_1. The induced subgraph is acyclic.

- If vertices $l_{1,2}$ and $l_{1,3}$ are chosen, while $l_{1,1}$ is not, the algorithm will choose vertices $b_{1,1}, b_{1,2}, b_{1,3}, b_{1,4}, b_{1,5}, b_{1,6}, b_{1,7}, b_{1,9}, b_{1,10}, b_{1,13}$ and o_1. The induced subgraph is acyclic.

- If vertices $l_{1,1}, l_{1,2}$ and $l_{1,3}$ are chosen, the algorithm will choose vertices $b_{1,1}, b_{1,2}, b_{1,3}, b_{1,5}, b_{1,6}, b_{1,7}, b_{1,9}, b_{1,10}, b_{1,13}$ and o_1. The induced subgraph is acyclic.

- In A_1, if vertex o_1 is chosen, while o_2 is not, the algorithm will choose $e_1, e_2, e_4, e_5, e_6, e_7, e_8, e_9, e_{11}, e_{12}, e_{13}$ and e_{14}. The induced subgraph is acyclic.

- If vertex o_2 is chosen, while o_1 is not then the algorithm will choose $e_1, e_2, e_3, e_4, e_5, e_7, e_8, e_{10}, e_{11}, e_{12}$ and e_{13} and e_{14}. The induced subgraph is acyclic.

- If both vertex o_1 and o_2 are chosen then the algorithm will subsequently choose $e_1, e_2, e_4, e_5, e_7, e_8, e_9, e_{10}, e_{11}, e_{13}, e_{14}$ and a_1. Again the induced subgraph is acyclic.

Similar reasoning shows that vertex t will be chosen if, and only if, all vertices o_i, where $1 \leq i \leq m$, have been chosen. Therefore t is chosen if, and only if, the truth assignment corresponding to the chosen path in P satisfies all the clauses in C. Vice versa, if a truth assignment satisfying all the clauses in C exists then the corresponding path will be chosen by an execution of GREEDY(bipartite) on instance (G, P, s); and the output of the execution will contain vertex t. The result follows. $\square$

To show that the problem GREEDY(partial ordering, maximum degree 2, bipartite) is solvable in polynomial time, we simply point out that, as G has maximum degree 2, vertex t can be in at most one cycle of odd length. To decide whether (G, P, s, t) is a yes-instance it is sufficient to check if all vertices on such a cycle appear in every path from s to t in P.

4.3.4 Planar

Lemma 4.4 *The problem GREEDY(partial ordering, bipartite with maximum degree* 3*, planar) is* **NP***-complete.*

Proof We will start the reduction with some graph theoretical definitions. Two graphs are *homeomorphic* if they can both be obtained from the same graph by a sequence of subdivisions of edges. For example, any 2 cycles are homeomorphic [21]. To prove this result we will use Kuratowski's Theorem.

Theorem 4.5 (Kuratowski's Theorem) A graph G is planar if, and only if, no subgraph of G is homeomorphic to $K_{3,3}$ or K_5.

We will use the fact that a graph homeomorphic to the complete bipartite graph $K_{3,3}$ is not planar. We describe the gadgets with reference to Figure 4.6. Note that the gadgets A_1 and A_i, for $2 \leq i \leq m-1$, are connected

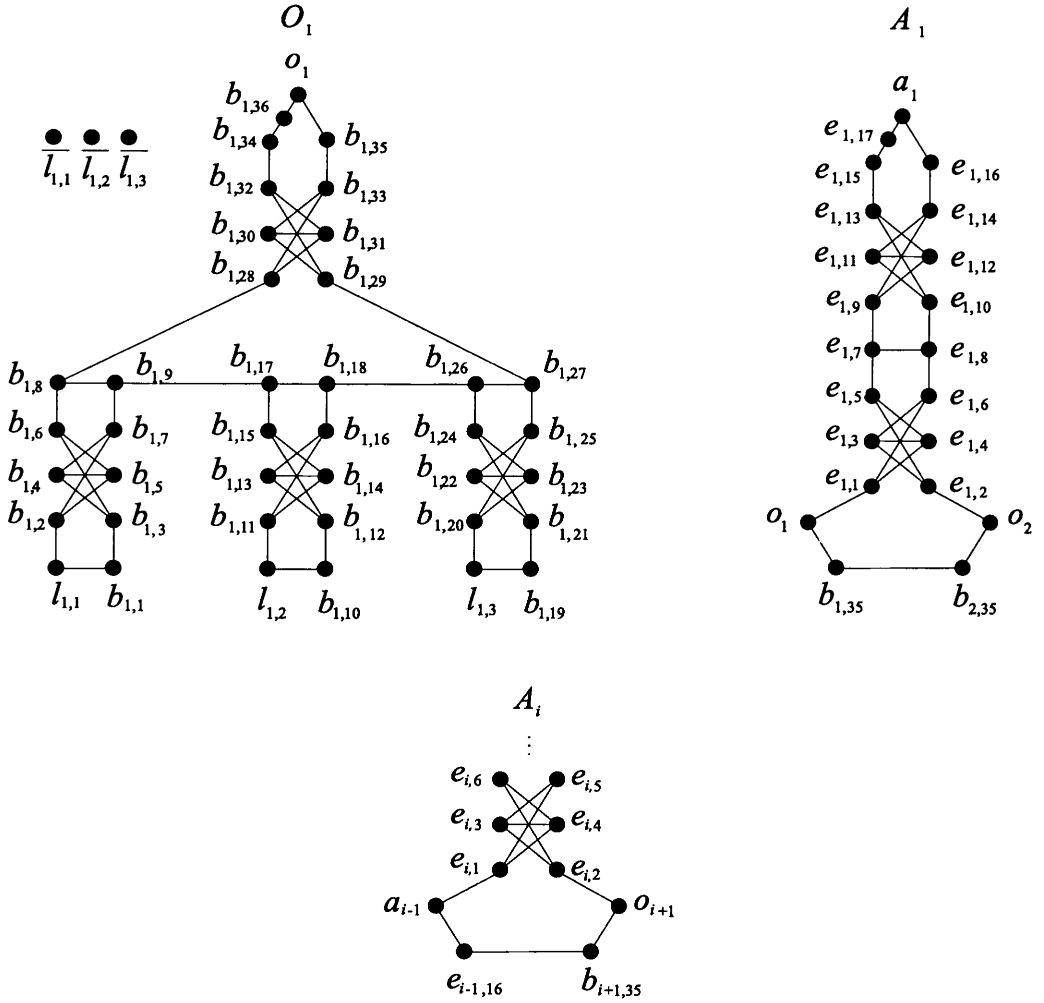


Figure 4.6: O_1, A_1 and A_i for property planar.

to the skeleton slightly differently. In A_1 b-vertex $b_{1,35}$ from O_1 is joined to b-vertex $b_{2,35}$ from O_2. For all other A gadgets A_i, for $2 \leq i \leq m-1$, vertex $e_{i-1,16}$, from A_{i-1} is joined to b-vertex $b_{i+1,35}$ from O_{i+1}.

In O_1, if vertex $l_{1,1}$ is chosen then $\{l_{1,1}, b_{1,1}, b_{1,2}, \ldots, b_{1,8}\}$ induce a planar subgraph of G, and therefore all vertices in the set will be chosen. The subgraph of G induced by $\{l_{1,1}, b_{1,1}, b_{1,2}, \ldots, b_{1,9}\}$, on the other hand, is homeomorphic to $K_{3,3}$ and therefore $b_{1,9}$ will be rejected. For the same reason, if vertex $l_{1,2}$ is chosen then vertices $b_{1,10}, b_{1,11}, \ldots, b_{1,17}$ will be chosen, but $b_{1,18}$ will not. If vertex $l_{1,3}$ is chosen, vertices $b_{1,19}, b_{1,20}, \ldots, b_{1,26}$ will be selected, but $b_{1,27}$ will be rejected. If none of $l_{1,1}, l_{1,2}, l_{1,3}$ is chosen, then $b_{1,9}, b_{1,18}$ and $b_{1,27}$ will be selected.

The execution of the algorithm proceeds by visiting vertices $b_{1,28}$ to $b_{1,36}$, and they will all be chosen. If any of the vertices $b_{1,9}, b_{1,18}, b_{1,27}$ have been rejected then vertex o_1 will be chosen, as the induced subgraph is still planar; while if they have all been chosen, o_1 will be rejected, as the subgraph of G induced by $\{b_{1,8}, b_{1,9}, b_{1,17}, b_{1,18}, b_{1,26}, b_{1,27}, \ldots, b_{1,36}, o_1\}$ is homeomorphic to $K_{3,3}$. It follows that o_1 will be chosen if, and only if, at least one of $l_{1,1}, l_{1,2}, l_{1,3}$ is selected.

In A_1, vertices $e_{1,1}, e_{1,2}, \ldots, e_{1,7}$, are chosen in every execution of the algorithm. Vertices $b_{1,35}$ and $b_{2,35}$ from O_1 and O_2, respectively, are also always chosen. If at least one of o_1, o_2 was rejected then vertex $e_{1,8}$ will be selected, else, as the subgraph of G induced by $\{b_{1,35}, b_{2,35}, o_1, o_2, e_{1,1}, e_{1,2}, \ldots, e_{1,8}\}$ is homeomorphic to $K_{3,3}$, $e_{1,8}$ will be rejected. The algorithm proceeds by visiting vertices $e_{1,9}, e_{1,10}, \ldots, e_{1,17}$, and they will all be chosen. If $e_{1,8}$ was previously rejected, that is, if both o_1 and o_2 have been chosen, then vertex a_1 will be chosen, else it will be rejected. The choice and rejection of vertices in the remaining gadgets A_i, where $2 \leq i \leq m-1$, is almost identical. The only difference is that vertices $e_{i-1,16}$ from A_{i-1} and $b_{i+1,35}$ from O_{i+1} are always chosen, and vertex a_i is chosen if, and only if, both vertex a_{i-1} and

o_{i+1} are chosen. Similar reasoning shows that vertex t will be chosen if, and only if, all vertices o_i, where $1 \leq i \leq m$, have been chosen. Therefore t is chosen if, and only if, the truth assignment corresponding to the chosen path in P satisfies all the clauses in C. Vice versa, if a truth assignment satisfying all the clauses in C exists then the corresponding path will be chosen by an execution of GREEDY(planar) on instance (G, P, s); and the output of the execution will contain vertex t.

The graph obtained by joining the copies of the gadgets is bipartite with maximum degree 3. To see that the graph is bipartite, we can think of a bipartite graph as a graph which is 2-colourable. As the gadgets O and A are bipartite, any 2 adjacent vertices must be coloured with 2 different colours. Whenever two copies of the gadgets are joined, this is done by identifying two adjacent vertices in one gadgets with two adjacent vertices in the other. It is clear that any proper 2-colouring on one gadget can be expanded to a proper colouring on the graph resulting from the union, and therefore such a graph must be bipartite as well. □

To show that the problem GREEDY(partial ordering, maximum degree 2, planar) is solvable in polynomial time, we simply point out that any graph of maximum degree 2 cannot be homeomorphic to $K_{3,3}$ or K_5, and is therefore planar.

4.3.5 Outerplanar

Lemma 4.6 *The problem GREEDY(partial ordering, planar and bipartite with maximum degree* 3, *outerplanar) is* **NP***-complete.*

Proof To prove this result we will use a theorem from [21] (stated below). An *outerplanar graph* is a planar graph that can be drawn with all its vertices on the same face, which is generally chosen to be the exterior face. We point out that the graph does not have to be connected, and that therefore all independent sets of vertices are outerplanar.

Theorem 4.7 A graph G is outerplanar if, and only if, every subgraph of G is either isomorphic to $K_4 - x$ (where x is any edge), or is not homeomorphic to K_4 or $K_{2,3}$.

We use the fact that any cycle of length ≥ 4 which contains a path (not on the cycle) of length ≥ 2 between any two nonadjacent cycle nodes, is homeomorphic to $K_{2,3}$, and use it as a forbidden graph for outerplanarity.

We describe the gadgets with reference to Figure 4.7. Note that the gadgets A_1 and A_i, for $2 \leq i \leq m-1$, are connected to the skeleton slightly differently. In A_1 b-vertex $b_{1,26}$ from O_1 is joined to b-vertex $b_{2,26}$ from O_2. For all other A gadgets A_i, for $2 \leq i \leq m-1$, vertex $e_{i-1,11}$, from A_{i-1} is joined to b-vertex $b_{i+1,26}$ from O_{i+1}.

In gadget O_1, if vertex $l_{1,1}$ is chosen then the subgraph induced by $\{l_{1,1}, b_{1,1}, b_{1,2}, \ldots, b_{1,6}\}$ is outerplanar, and therefore all the vertices in the set will be selected. The addition of vertex $b_{1,7}$, on the other hand, would induce a subgraph of G homeomorphic to $K_{2,3}$, and $b_{1,7}$ will therefore be rejected. If $l_{1,1}$ is not chosen then $b_{1,7}$ is selected. Using the same sort of reasoning, it is not difficult to see that if vertex $l_{1,2}$ is chosen then vertices $b_{1,8}, b_{1,9}, \ldots, b_{1,13}$ will be selected, while $b_{1,14}$ will be rejected. If $l_{1,2}$ is not

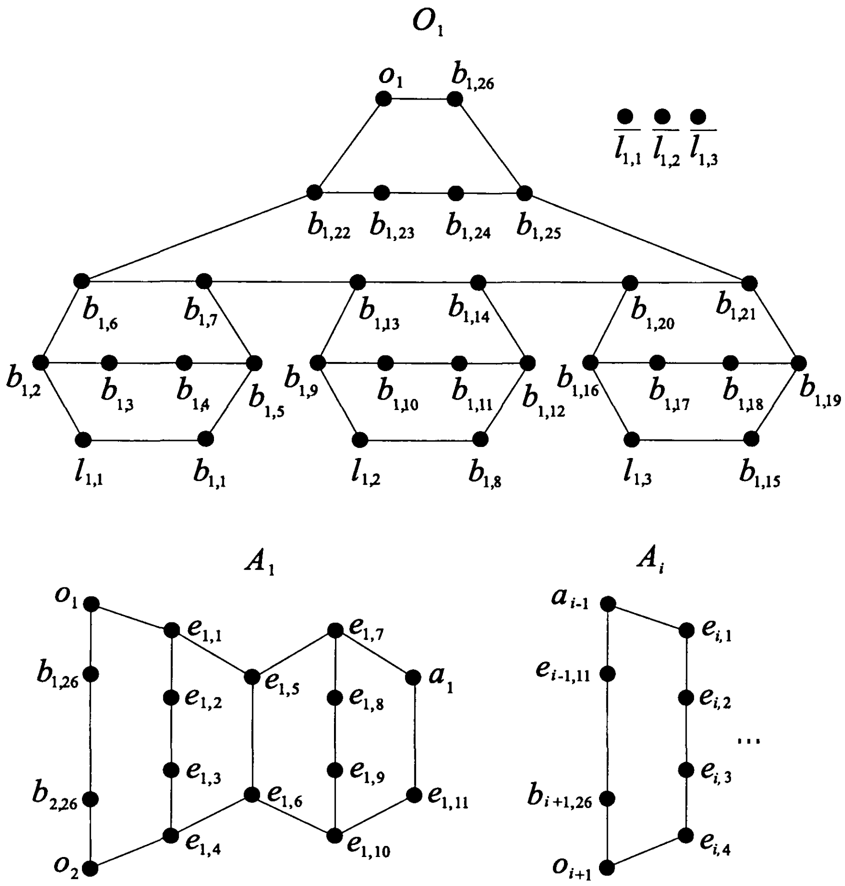


Figure 4.7: O_1, A_1 and A_i for property outerplanar.

chosen then $b_{1,14}$ is added to the set of selected vertices. If $l_{1,3}$ is chosen then vertex $b_{1,21}$ is rejected, else it is chosen. Vertices $b_{1,15}, b_{1,16}, \ldots, b_{1,20}$ are always chosen.

The algorithm proceeds by examining vertices $b_{1,22}, b_{1,23}, \ldots, b_{1,26}$. Regardless of whether $l_{1,1}, l_{1,2}$ and $l_{1,3}$ are chosen or rejected, the addition of vertices $b_{1,22}, b_{1,23}, \ldots, b_{1,26}$ to the set of chosen vertices does not result in a graph homeomorphic to $K_{2,3}$ (or K_4), therefore all such vertices will be chosen by any run of the algorithm. If $b_{1,7}, b_{1,14}$ and $b_{1,21}$ have all been chosen then o_1 will be rejected, as choosing it would induce a subgraph of G homeomorphic to $K_{2,3}$. If at least one of $b_{1,7}, b_{1,14}, b_{1,21}$ was rejected, which means that at least one of $l_{1,1}, l_{1,2}, l_{1,3}$ was selected, then vertex o_1 will be chosen.

When considering gadget A_1, vertices $b_{1,26}$ and $b_{2,26}$, from O_1 and O_2 respectively, are always chosen. If o_1 and o_2 are selected then the algorithm will choose $e_{1,1}, e_{1,2}, \ldots, e_{1,5}$, but it will reject $e_{1,6}$, as the subgraph of G induced by $\{o_1, o_2, b_{1,26}, b_{2,26}, e_{1,1}, e_{1,2}, \ldots, e_{1,6}\}$ is homeomorphic to $K_{2,3}$. If $e_{1,6}$ is rejected then vertices $e_{1,7}, e_{1,8}, \ldots, e_{1,11}$ will be chosen, and a_1 will be selected as well. If at least one of o_1, o_2 is rejected then $e_{1,6}$ will be chosen, and this will result in vertex a_1 being rejected. It follows that a_1 will be chosen if, and only if, both o_1 and o_2 are selected. The choice and rejection of vertices in the remaining gadgets A_i, where $2 \leq i \leq m-1$, is almost identical. The only difference is that vertices $e_{i-1,11}$ from A_{i-1} and $b_{i+1,26}$ from O_{i+1} are always chosen, and vertex a_i is chosen if, and only if, both vertex a_{i-1} and o_{i+1} are chosen. Similar reasoning results in vertex t being chosen if, and only if, all vertices o_i, where $1 \leq i \leq m$, have been chosen. Therefore t is chosen if, and only if, the truth assignment corresponding to the chosen path in P satisfies all the clauses in C. Vice versa, if a truth assignment satisfying all the clauses in C exists, then similar reasoning yields that the corresponding path will be chosen by an execution of GREEDY(outerplanar) on instance (G, P, s); and the output of the execution will contain vertex t.

It is straightforward to notice that graph G is planar, bipartite and has maximum degree 3. $\square$

To show that the problem GREEDY(partial ordering, maximum degree 2, outerplanar) is solvable in polynomial time, it is sufficient to point out that any graph of degree 2 or less is outerplanar because it cannot be homeomorphic to $K_{2,3}$ or K_4.

4.3.6 Edge graph

Given a graph $G = (V, E)$, the corresponding *edge graph* $L(G) = (E, D)$ is the graph that has as vertex set the edge set of G, and such that 2 vertices in $L(G)$ are adjacent if, and only if, the corresponding edges in G have a vertex in common. We say that a graph G is an edge graph if there exists a graph T such that G is isomorphic to the edge graph $L(T)$ of T. It is possible to describe this property in terms of forbidden subgraphs [21].

Theorem 4.8 A graph is an edge graph if, and only if, it does not contain any of the 9 graphs in Figure 4.8 as an induced subgraph.

Lemma 4.9 *The problem GREEDY(partial ordering, planar and bipartite with maximum degree* 3*, edge graph) is* **NP***-complete.*

We describe the gadgets with reference to Figure 4.9.

In gadget O_1, if vertex $l_{1,1}$ is chosen then $b_{1,1}$ and $b_{1,2}$ will be chosen, but $b_{1,3}$ will be rejected as the subgraph induced by $\{l_{1,1}, b_{1,1}, b_{1,2}, b_{1,3}\}$ is one of the forbidden subgraphs shown in Figure 4.8 (G_1). If $l_{1,1}$ is not chosen then $b_{1,3}$ will be selected. For the same reason, if vertex $l_{1,2}$ is chosen then vertices $b_{1,4}$ and $b_{1,5}$ will be selected, but $b_{1,6}$ will be rejected. If $l_{1,2}$ is not chosen then

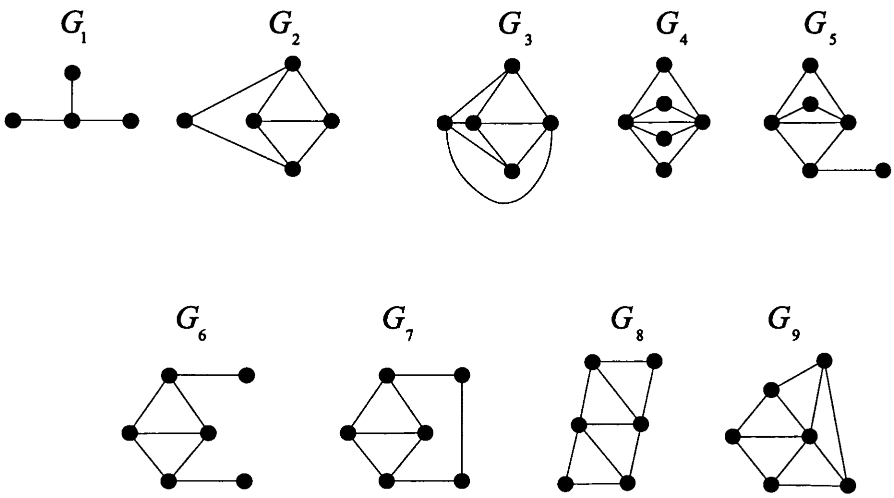


Figure 4.8: Forbidden graphs for property edge graph.

Proof

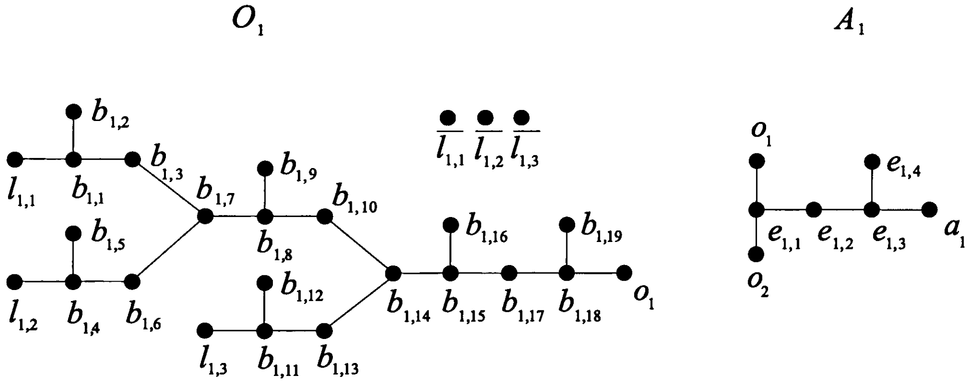


Figure 4.9: O_1 and A_1 for property edge graph.

$b_{1,6}$ will be selected. If $l_{1,3}$ is chosen then vertices $b_{1,11}$ and $b_{1,12}$ are chosen, but $b_{1,13}$ is rejected. If $l_{1,3}$ is not chosen then $b_{1,13}$ is selected. Vertex $b_{1,7}$ is chosen by every execution of the algorithm. If at least one of $b_{1,3}, b_{1,6}$ was rejected, that is, if at least one of $l_{1,1}$ and $l_{1,2}$ was previously chosen, then vertex $b_{1,8}$ will be selected. Vertex $b_{1,9}$ will then be chosen, and $b_{1,10}$ will be rejected. If vertex $b_{1,8}$ was rejected then $b_{1,10}$ will be chosen. Vertex $b_{1,14}$ is always chosen. If both vertices $b_{1,10}$ and $b_{1,13}$ are chosen, that is, none of $l_{1,1}, l_{1,2}, l_{1,3}$, was selected, then $b_{1,15}$ will be rejected, while $b_{1,17}$ will be chosen; this means that vertex o_1 will be rejected. If either $b_{1,10}$ or $b_{1,13}$ are rejected, which happens if, and only if, at least one of $l_{1,1}, l_{1,2}, l_{1,3}$ was chosen, then $b_{1,15}$ and $b_{1,16}$ will be chosen, vertex $b_{1,17}$ will be rejected, while $b_{1,18}, b_{1,19}$ and o_1 will be chosen.

In gadget A_1, if both o_1 and o_2 are chosen then vertex $e_{1,1}$ is selected, but $e_{1,2}$ is rejected, which in turn means that $e_{1,3}, e_{1,4}$ and a_1 will be chosen. If at most one of o_1, o_2 is chosen then vertices $e_{1,1}, e_{1,2}, e_{1,3}, e_{1,4}$ will be selected, but vertex a_1 will be rejected. Similar reasoning shows that vertex t will be chosen if, and only if, all vertices o_i, where $1 \leq i \leq m$, have been chosen. Therefore t is chosen if, and only if, the truth assignment corresponding to the chosen path in P satisfies all the clauses in C. Vice versa, if a truth assignment satisfying all the clauses in C exists then the corresponding path will be chosen by an execution of GREEDY(edge graph) on instance (G, P, s); and the output of the execution will contain vertex t.

It is not difficult to see that the graph formed by the union of the gadgets is planar, bipartite and has maximum degree 3. $\square$

To show that the problem GREEDY(partial ordering, maximum degree 2, edge graph) is solvable in polynomial time, it is sufficient to point out that all forbidden subgraphs shown in Figure 4.8 have degree ≥ 3, and therefore all graphs of degree ≤ 2 are edge graphs.

4.3.7 Interval graph

Lemma 4.10 *The problem GREEDY(partial ordering, planar and bipartite with maximum degree* 3, *interval graph) is* **NP***-complete.*

Proof An *interval graph* is a graph such that there exists a set of intervals on the real line in a one-to-one correspondence with the vertices of the graph. Two vertices are adjacent if, and only if, their corresponding intervals intersect. The property can be described in terms of forbidden subgraphs by using the following theorem [26].

Theorem 4.11 *A graph G is an interval graph if, and only if, it does not contain any of the graphs shown in Figure* 4.10 *as a subgraph.*

We describe the gadgets with reference to Figure 4.11.

In gadget O_1, vertices $b_{1,1}$ and $b_{1,2}$ are always chosen. If vertex $l_{1,1}$ was previously selected then $b_{1,3}$ will be rejected, as the subgraph induced by $\{l_{1,1}, b_{1,1}, b_{1,2}, b_{1,3}\}$ is a 4-cycle, which is one of the forbidden graphs (G_3) shown in Figure 4.10. If $l_{1,1}$ is not chosen then $b_{1,3}$ will be selected. Vertices $b_{1,4}$ and $b_{1,5}$ are chosen next. When $b_{1,6}$ is examined, it will be chosen if $l_{1,2}$ was previously rejected, and it will be rejected otherwise. Vertices $b_{1,7}$ and

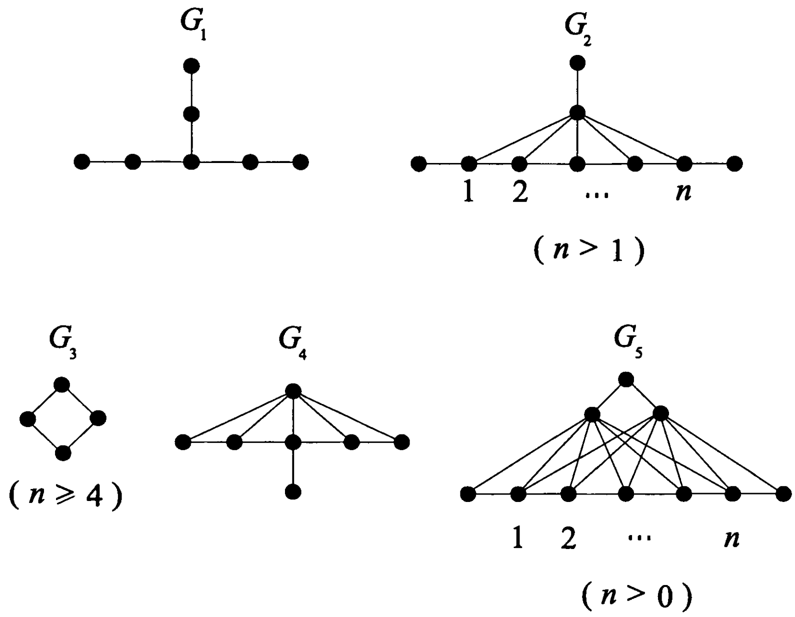


Figure 4.10: Forbidden graphs for property interval graph.

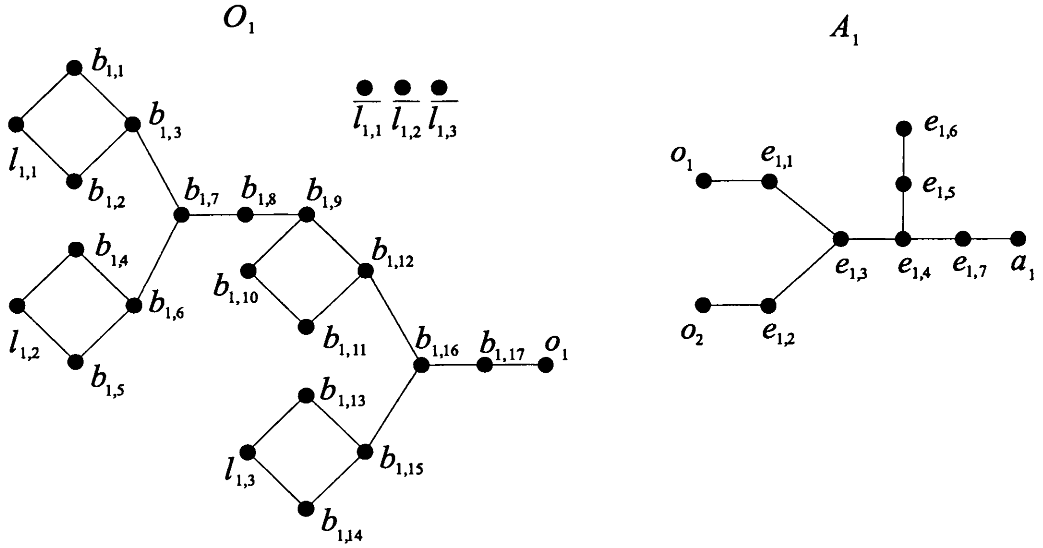


Figure 4.11: O_1 and A_1 for property interval graph.

$b_{1,8}$ are always selected. Vertex $b_{1,9}$ is chosen if at least one of $b_{1,3}$ and $b_{1,6}$ was rejected, which means that at least one of $l_{1,1}, l_{1,2}$ was chosen. This is because the subgraph induced by $\{b_{1,1}, b_{1,3}, b_{1,7}, b_{1,6}, b_{1,5}, b_{1,8}, b_{1,9}\}$ is one of the forbidden graphs (G_1), shown in Figure 4.10. Vertices $b_{1,10}$ and $b_{1,11}$ are examined next, and they are chosen by every execution of the algorithm, while $b_{1,12}$ will be selected if, and only if, $b_{1,9}$ was rejected. Vertices $b_{1,13}$ and $b_{1,14}$ are always selected, while $b_{1,15}$ is chosen only if $l_{1,3}$ is rejected. Vertices $b_{1,16}$ and $b_{1,17}$ are always selected, and o_1 is chosen if, and only if, at least one of $b_{1,12}, b_{1,15}$ was rejected. It follows that o_1 is chosen if, and only if, at least one of $l_{1,1}, l_{1,2}, l_{1,3}$ is chosen.

In the gadget A_1, if both vertex o_1 and o_2 are selected, then vertices $e_{1,1}, e_{1,2}, e_{1,3}$ and $e_{1,4}$ will be chosen, but vertex $e_{1,5}$ will be rejected. Vertex $e_{1,6}$ will then be chosen, while $e_{1,7}$ will be rejected. Vertex a_1 will then be chosen. If at least one of o_1 and o_2 is not chosen then vertices $e_{1,1}, e_{1,2}, \ldots, e_{1,7}$ will be selected and vertex a_1 will therefore be rejected. Similar reasoning results in vertex t being chosen if, and only if, all vertices o_i, where $1 \leq i \leq m$, have been chosen. Therefore t is chosen if, and only if, the truth assignment corresponding to the chosen path in P satisfies all the clauses in C. Vice versa, similar reasoning yields that if a truth assignment satisfying all the clauses in C exists then the corresponding path will be chosen by an execution of GREEDY(interval graph) on instance (G, P, s); and the output of the execution will contain vertex t.

Graph G is planar, bipartite and has maximum degree 3. □

To show that 3 is the optimal degree bound for the property "interval graph", we will show that the problem GREEDY(partial ordering, maximum degree 2, interval graph) can be solved in polynomial time. Let us consider an instance (G, P, s, t) of the problem. The forbidden graphs for the property "interval graph" all have degree ≥ 3 apart from G_3, which is a cycle of length ≥ 4. As the maximum degree of any vertex in G is 2, it follows that vertex t in G can be in at most one cycle. We can determine in polynomial time if this is the case, and, if t is in a cycle of length ≥ 4, which vertices lie on the cycle. Like in the case of the property "k-cycle free", by determining whether it is possible to reach vertex t from s in $P \setminus \{x\}$, where x is any vertex on the cycle containing t, we can determine if (G, P, s, t) is a yes-instance or a no-instance of the problem in polynomial time.

4.3.8 Acyclic

Lemma 4.12 *The problem GREEDY(partial ordering, planar and bipartite with maximum degree* 3*, acyclic) is* **NP***-complete.*

Proof We describe the gadgets with reference to Figure 4.12. Note that the gadgets A_1 and A_i, for $2 \leq i \leq m-1$, are connected to the skeleton slightly differently. In A_1 e-vertex $e_{1,1}$ is adjacent to b-vertex $b_{1,19}$ from O_1. For all other A gadgets A_i, for $2 \leq i \leq m-1$, vertex $e_{i,1}$ is adjacent to vertex $e_{i-1,20}$ from A_{i-1}.

In the gadget O_1, if $l_{1,1}$ is chosen then $b_{1,1}, b_{1,2}, b_{1,3}, b_{1,4}$ will be chosen, but vertex $b_{1,5}$ will be rejected, as the subgraph of G induced by $\{l_{1,1}, b_{1,1}, \ldots, b_{1,5}\}$ is a cycle. If $l_{1,1}$ is not chosen then vertex $b_{1,5}$ will be selected. The next

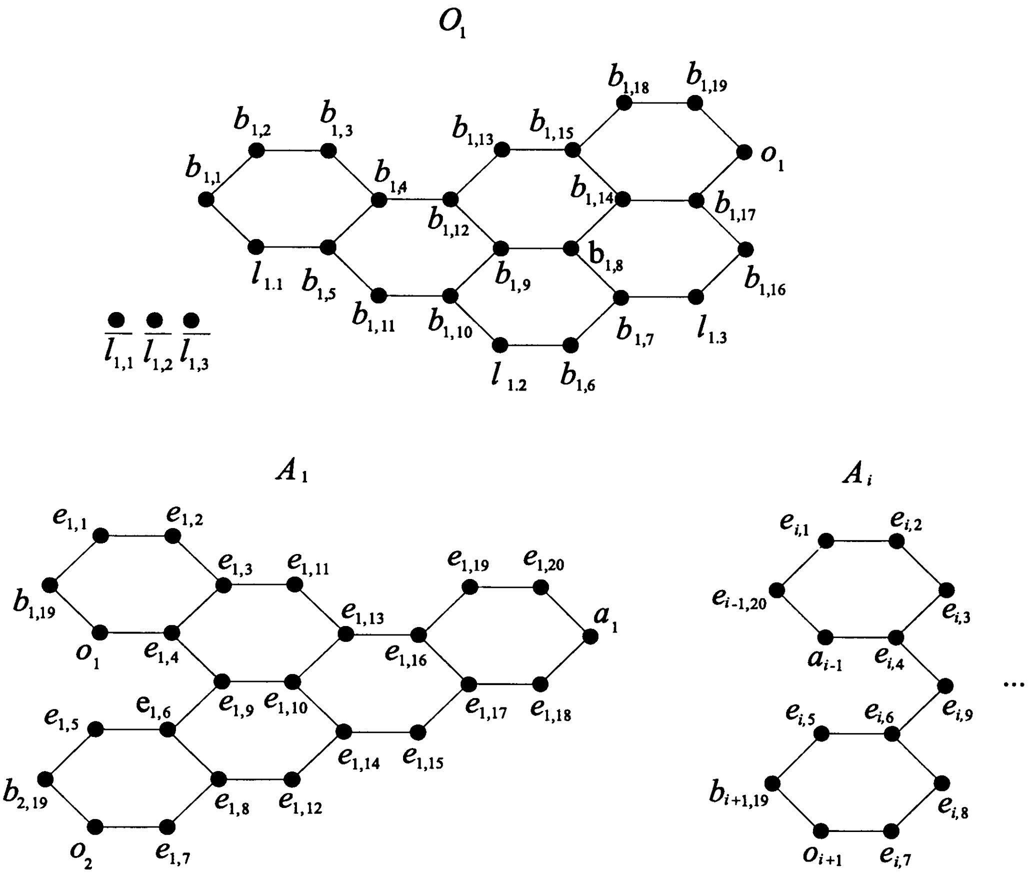


Figure 4.12: O_1, A_1 and A_i for property acyclic.

vertices to be examined will be $b_{1,6}, b_{1,7}, b_{1,8}$ and $b_{1,9}$, and they will all be selected. If vertex $l_{1,2}$ was previously selected then vertex $b_{1,10}$ will be rejected, if $l_{1,2}$ was not chosen then $b_{1,10}$ will be selected. The next vertex to be chosen is $b_{1,11}$, and the algorithm will proceed by examining $b_{1,12}$. If both vertex $b_{1,5}$ and $b_{1,10}$ were chosen, which means that none of $l_{1,1}$ and $l_{1,2}$ were selected, then $b_{1,12}$ will be rejected. If at least one of $l_{1,1}$ and $l_{1,2}$ was chosen then $b_{1,12}$ will be selected. Then vertices $b_{1,13}$ and $b_{1,14}$ will be chosen, and $b_{1,15}$ will be selected if, and only if, $b_{1,12}$ was rejected. Vertex $b_{1,16}$ is examined next, and it is always chosen; as vertices $b_{1,7}, b_{1,8}$ and $b_{1,14}$ were previously chosen, it follows that $b_{1,17}$ will be selected if, and only if, $l_{1,3}$ is not chosen. Then the algorithm examines $b_{1,18}$ and $b_{1,19}$, and they are always chosen. If at least one of $b_{1,15}$ and $b_{1,17}$ have been rejected, that is, at least one of $l_{1,1}, l_{1,2}, l_{1,3}$ was chosen then o_1 will be selected, else it will be rejected.

In gadget A_1 vertices $b_{1,19}$ and $b_{2,19}$, from O_1 and O_2 respectively, are always chosen. If o_1 is selected then vertices $e_{1,1}, e_{1,2}$ and $e_{1,3}$ will be chosen, but $e_{1,4}$ will be rejected, or the induced subgraph would contain a cycle. If o_1 is rejected then $e_{1,4}$ will be chosen. If o_2 is selected then vertices $e_{1,5}, e_{1,6}, e_{1,7}$ will be chosen, but $e_{1,8}$ will be rejected. If o_2 is not chosen then $e_{1,8}$ will be selected. Vertices $e_{1,9}, e_{1,10}, e_{1,11}$ and $e_{1,12}$ will be examined next, and they will all be chosen. The execution of the algorithm proceeds by visiting $e_{1,13}$, which will be chosen if, and only if, $e_{1,4}$ was rejected, that is, if o_1 was chosen. Vertex $e_{1,14}$ is chosen if, and only if, $e_{1,8}$ was rejected, that is, if o_2 was chosen. Then vertices $e_{1,15}$ and $e_{1,16}$ are chosen. Vertex $e_{1,17}$ is examined next, and it will be rejected if, and only if, both $e_{1,13}$ and $e_{1,14}$ were previously chosen. If $e_{1,17}$ is rejected, this will result in a_1 being chosen, while if $e_{1,17}$ is selected, then a_1 will be rejected. It follows that a_1 will be chosen if, and only if, both o_1 and o_2 are chosen. The choice and rejection of vertices in the remaining gadgets A_i, where $2 \leq i \leq m - 1$, is almost identical. The only difference is that vertices $e_{i-1,20}$ from A_{i-1} and $b_{i+1,19}$ from O_{i+1} are always chosen, and vertex a_i is chosen if, and only if, both vertex a_{i-1} and o_{i+1} are chosen. Similar reasoning results in vertex t being chosen if, and only if, all vertices o_i, where $1 \leq i \leq m$, have been chosen. Therefore t is chosen if, and only if, the truth assignment corresponding to the chosen path in P satisfies all the clauses in C. Vice versa, if a truth assignment satisfying all the clauses in C exists, then similar reasoning yields that the corresponding path will be chosen by an execution of GREEDY(acyclic) on instance (G, P, s); and the output of the execution will contain vertex t.

The graph obtained by the union of the gadgets is planar, bipartite, and with maximum degree 3. □

The degree bound 3 is optimal for this property because the problem GREEDY(partial ordering, maximum degree 2, acyclic) can be solved in polynomial time. Given an instance (G, P, s, t) of the problem, we can determine whether it is a yes- or a no-instance by simply checking if there exists a path in P from s to t that does not visit all vertices in the (at most) one cycle that contains vertex t in G. If such a path exists then (G, P, s, t) is a yes-instance of the problem, else it is a no-instance.

4.3.9 Chordal

Lemma 4.13 *The problem GREEDY(partial ordering, planar and bipartite with maximum degree* 3, *chordal) is* **NP***-complete.*

Proof The result follows from the proof of property "acyclic", because the same gadgets can be used to prove property "chordal". We just point out that we take as our forbidden graph a cycle of length 6, like we did in the previous section, and stress the fact that any subgraph of a graph, in which the smallest cycle has length 6, is acyclic if, and only if, it is chordal. □

We will now show that the problem GREEDY(partial ordering, maximum degree 2, chordal) is solvable in polynomial time. Note that every chordal graph G is such that every cycle in G of length ≥ 4 has a chord. As our instance graph has maximum degree 2, it follows that every vertex can be in at most one cycle. If vertex t appears on a cycle of length at least 4, we proceed like we did in the case of property acyclic and check if there is a path in P that does not visit all the vertices on the cycle before reaching t.

4.4 Near optimal degree bounds

In this section we consider properties for which the best degree bound we have obtained might not be optimal. We have not been able to find the exact boundary between **NP**-completeness and tractability, but we know that by restricting the degree bound on the instance graph by two more units we obtain a problem solvable in polynomial time. The corresponding results are summarised in Table 4.1, numbers 10-12, on page 60.

4.4.1 4-cycle free

Lemma 4.14 *The problem GREEDY(partial ordering, planar and bipartite with maximum degree* 4, 4*-cycle free) is* **NP***-complete.*

Proof We describe the gadgets with reference to Figure 4.13.

In gadget O_1, if vertex $l_{1,1}$ is chosen then vertices $b_{1,1}$ and $b_{1,2}$ will be chosen, but $b_{1,3}$ will be rejected, because the subgraph of G induced by

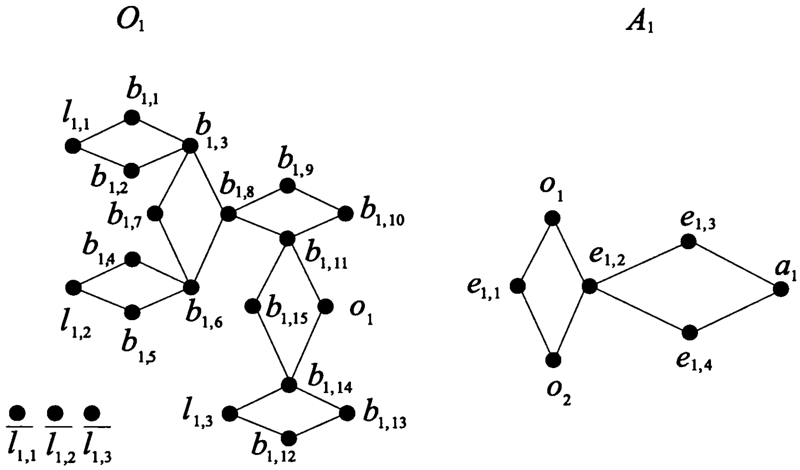


Figure 4.13: O_1 and A_1 for property 4-cycle free.

$\{l_{1,1}, b_{1,1}, b_{1,2}, b_{1,3}\}$ is a 4-cycle. If $l_{1,1}$ is not chosen then $b_{1,3}$ will be selected. If $l_{1,2}$ is chosen, then vertices $b_{1,4}$ and $b_{1,5}$ will be selected, but $b_{1,6}$ will be rejected. If $l_{1,2}$ is not chosen then vertex $b_{1,6}$ will be selected. Vertex $b_{1,7}$ is always chosen. If at least one of $b_{1,3}$ and $b_{1,6}$ was rejected, that is, if at least one of $l_{1,1}$ and $l_{1,2}$ was chosen then $b_{1,8}$ will be selected, else it will be rejected. Vertices $b_{1,9}$ and $b_{1,10}$ will be examined next, and they will be chosen by every execution of the algorithm. Vertex $b_{1,11}$ will be selected if, and only if, vertex $b_{1,8}$ was rejected. If $l_{1,3}$ was chosen then vertices $b_{1,12}$ and $b_{1,13}$ will be selected, but $b_{1,14}$ will not. If $l_{1,3}$ was rejected then $b_{1,14}$ will be chosen. Vertex $b_{1,15}$ is examined next, and it is always chosen. If at least one of $b_{1,11}$ and $b_{1,14}$ was rejected, that is, if at least one of $l_{1,1}, l_{1,2}, l_{1,3}$ was chosen, then vertex o_1 will be selected, else it will be rejected.

In the gadget A_1, vertex $e_{1,1}$ is always selected. If both o_1 and o_2 are selected then vertex $e_{1,2}$ will be rejected, which means that a_1 will be chosen.

If at least one of o_1, o_2 was rejected then $e_{1,2}, e_{1,3}$ and $e_{1,4}$ will be chosen, but a_1 will be rejected. Similar reasoning shows that vertex t will be chosen if, and only if, all vertices o_i, where $1 \leq i \leq m$, have been chosen. Therefore t is chosen if, and only if, the truth assignment corresponding to the chosen path in P satisfies all the clauses in C. Vice versa, if a truth assignment satisfying all the clauses in C exists then the corresponding path will be chosen by an execution of GREEDY(4-cycle free) on instance (G, P, s); and the output of the execution will contain vertex t.

It is straightforward to notice that G is planar, bipartite and each vertex has degree at most 4. □

We do not know the complexity of the problem GREEDY(partial ordering, maximum degree 3, 4-cycle free) but we know that the problem is solvable in polynomial time if we restrict the instance graph to have degree at most 2. To see this, consider an instance (G, P, s, t). As the degree bound is 2, vertex t can be in at most one 4-cycle. By checking whether there exists a path in P from s to t that does not contain all vertices on the (at most) one 4-cycle containing t in G, we can determine whether (G, P, s, t) is a yes- or a no-instance of our problem.

4.4.2 Maximum degree 1

Lemma 4.15 *The problem GREEDY(partial ordering, planar and bipartite with maximum degree* 3*, maximum degree* 1*) is* **NP***-complete.*

Proof We describe the gadgets with reference to Figure 4.14.

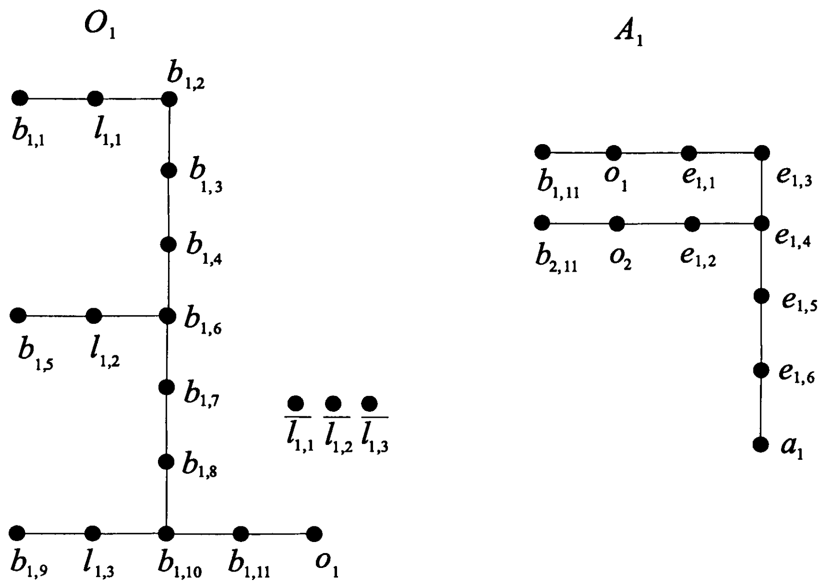


Figure 4.14: O_1 and A_1 for property maximum degree 1.

In the gadget O_1, vertices $b_{1,1}, b_{1,5}$ and $b_{1,9}$ are always chosen, as they have degree 1. If $l_{1,1}$ is chosen then $b_{1,2}$ will not be selected or, in the induced subgraph, $l_{1,1}$ would have degree 2. Then vertices $b_{1,3}$ and $b_{1,4}$ will be selected, vertex $b_{1,6}$ will be rejected, $b_{1,7}$ and $b_{1,8}$ will be chosen, and $b_{1,10}$ will be rejected. Vertex $b_{1,11}$, which is examined next, will be chosen and so will o_1. If vertex $l_{1,1}$ is not chosen then vertices $b_{1,2}$ and $b_{1,3}$ will be selected, while $b_{1,4}$ will be rejected. If vertex $l_{1,2}$ was chosen then vertex $b_{1,6}$ will be rejected, or otherwise $l_{1,2}$ would have degree 2 in the induced subgraph. The fact that $b_{1,6}$ is rejected results, as explained before, in o_1 being chosen. If $l_{1,2}$ is not chosen then $b_{1,6}$ will be selected; $b_{1,7}$ will be chosen, and therefore $b_{1,8}$ will be rejected. If vertex $l_{1,3}$ was chosen then $b_{1,10}$ will be rejected, and therefore o_1 will be chosen. If $l_{1,3}$ was not chosen then $b_{1,10}$ will be selected, and this results in o_1 being rejected. It follows that o_1 is chosen if, and only if, at least one of $l_{1,1}, l_{1,2}, l_{1,3}$ is selected.

Vertices $b_{1,11}$ and $b_{2,11}$ from O_1 and O_2 are always selected. In the gadget A_1, if both o_1 and o_2 are chosen then vertices $e_{1,1}$ and $e_{1,2}$ will be rejected, or in the induced subgraph o_1 and o_2 would have degree 2. The execution proceeds by visiting $e_{1,3}$ and $e_{1,4}$, and they will both be chosen. Vertex $e_{1,5}$ will be rejected, and therefore $e_{1,6}$ and a_1 will be chosen. If vertex o_1 is not chosen then $e_{1,1}$ will be selected, vertex $e_{1,3}$ will be chosen, but $e_{1,4}$ will be rejected, or otherwise $e_{1,3}$ would have degree 2 in the induced subgraph. Then vertices $e_{1,5}$ and $e_{1,6}$ will be chosen, and a_1 will be rejected. If vertex o_2 is not chosen then $e_{1,2}$ will be selected, and as $e_{1,3}$ is always chosen, vertex $e_{1,4}$ will be rejected. Again this results in a_1 being rejected. The choice and rejection of vertices in the remaining gadgets A_i, where $2 \leq i \leq m-1$, is almost identical. The only difference is that vertices $e_{i-1,6}$ from A_{i-1} and $b_{i+1,11}$ from O_{i+1} are always chosen, and vertex a_i is chosen if, and only if, both vertex a_{i-1} and o_{i+1} are chosen. Similar reasoning results in vertex t being chosen if, and only if, all vertices o_i, where $1 \leq i \leq m$, have been chosen. Therefore t is chosen if, and only if, the truth assignment corresponding to the chosen path in P satisfies all the clauses in C. Vice versa, similar reasoning yields that if a truth assignment satisfying all the clauses in C exists then the corresponding path will be chosen by an execution of GREEDY(maximum degree 1) on instance (G, P, s); and the output of the execution will contain vertex t.

The graph obtained by joining the gadgets is planar, bipartite and with maximum degree 3. $\square$

We do not know the complexity of the problem GREEDY(partial ordering, maximum degree 2, maximum degree 1) but it is straightforward to notice that the problem becomes trivial if the instance graph has maximum degree 1.

4.4.3 Independent set

Lemma 4.16 *The problem GREEDY(partial ordering, planar and bipartite with maximum degree* 3, *independent set) is* **NP***-complete.*

Proof The result follows from Theorem 3.6. □

We do not know the complexity of the problem GREEDY(partial ordering, maximum degree 2, independent set) but the problem becomes trivial if, in the instance graph, vertices have degree at most 1.

We conclude the section by showing, in Table 4.2, a summary of the degree bounds that we have found for which the problem GREEDY(partial order, $\mathcal{C}$, π) is solvable in polynomial time. The column *Property* indicates the property π under consideration. The columns *Restriction* and *Degree* characterise the class $\mathcal{C}$ of undirected graphs considered (the ordering is always a partial ordering). So, for example, the first row of the table indicates that the problem GREEDY(partial ordering, maximum degree 3, 3-cycle free) is solvable in polynomial time.

No.	Property	Restriction	Degree
(1)	3-cycle free	undirected graphs	3
(2)	k-cycle free $k \geq 5$	undirected graphs	2
(3)	bipartite	undirected graphs	2
(4)	planar	undirected graphs	2
(5)	outerplanar	undirected graphs	2
(6)	edge graph	undirected graphs	2
(7)	interval graph	undirected graphs	2
(8)	acyclic	undirected graphs	2
(9)	chordal	undirected graphs	2
(10)	4-cycle free	undirected graphs	2
(11)	maximum degree 1	undirected graphs	1
(12)	independent set	undirected graphs	1

Table 4.2: Polynomial time solvable problems

4.5 Conclusion

In this chapter we considered restrictions on the maximum degree of the vertices of the instance graph G for which the problem GREEDY(partial ordering, $\mathcal{C}$, π) remains **NP**-complete; we have also shown how it is possible to solve the problem in deterministic polynomial time if we restrict the degrees any further. The natural direction in which to extend the research would be to try to answer the question:

Can we obtain optimal degree bounds for properties: 4*-cycle free, independent set and maximum degree one?*

So far we have not been able to find an answer, and we leave this question open. Instead, we explore the complexity of the problem GREEDY(partial ordering, $\mathcal{C}$, π) when we consider properties π which are not testable in deterministic polynomial time, but are testable in nondeterministic polynomial time. This will be the topic of the next chapter.

Chapter 5

A complexity-theoretic dichotomy result

5.1 Introduction

In the previous two chapters we dealt with the problem GREEDY(partial ordering, $\mathcal{C}$, π) under the assumption that π is a hereditary property, non-trivial on $\mathcal{C}$ and testable in deterministic polynomial time. In this chapter we will examine the complexity of the problem if we drop the requirement that the property be testable in deterministic polynomial time. We remark here that the results of this chapter were presented in the paper by A. Puricella and I. A. Stewart, Greedy algorithms, H-colourings and a complexity-theoretic dichotomy, *Theoretical Computer Science*, to appear [34].

In what is now a seminal result, Hell and Nešetřil [23] established a dichotomy for the H-colouring problem when H is an undirected graph with no self-loops (if H contains a loop the problem becomes trivial as all vertices can be mapped to the vertex with the loop): the H-colouring problem is in **P**, if H is bipartite, and is **NP**-complete otherwise (notice that the existence of an H-colouring of an undirected graph G, *i.e.*, a homomorphism from G to H, is a particular hereditary property of G). Such a (dichotomy) result can also be thought of as a generic result in that it provides a complete, exact classification of the computational complexities of an infinite class of problems (in this case, the class of H-colouring problems).

A number of other dichotomy results (involving unequivocal complexity-theoretic classifications) and generic results (applicable to an infinite class of problems) have since been obtained. Some examples are: Feder and Hell's result [13] that the list homomorphism problem for reflexive graphs is solvable in polynomial time if the target graph is an interval graph, and **NP**-complete otherwise; Feder, Hell and Huang's [14] result that the list homomorphism problem for irreflexive graphs is solvable in polynomial time if the complement of the target graph is a circular arc graph of clique covering number two, and **NP**-complete otherwise; Díaz, Serna and Thilikos's result [10] that the complexity of the list (H, C, K)-colouring problem mirrors that of the list homomorphism problem; and Dyer and Greenhill's result [11] that the problem of counting the H-colourings of a graph is solvable in polynomial time if every connected component of H is a complete reflexive graph with all loops present or a complete bipartite irreflexive graph (with no loops present), and $\sharp$**P**-complete otherwise.

Dichotomy and generic results such as those highlighted above are particularly attractive as they give a concise and simplified view of a parameterised world of natural problems. In this chapter, we consider the problem of deciding whether a given vertex of a given undirected graph G, whose vertices are partially ordered, lies in a lexicographically first maximal H-colourable subgraph of G (where the undirected graph H is fixed). That is, we examine the complexity of the problem GREEDY(partial ordering, undirected graphs, H-colourable). In particular, we prove that this problem is **NP**-complete, if H is bipartite, and $\mathbf{\Sigma_2^p}$-complete, if H is non-bipartite; thus establishing yet another complexity-theoretic dichotomy result. Our proofs use the techniques established by Hell and Nešetřil in [23] although they are combinatorially adapted according to our circumstances. However, part of Hell and Nešetřil's constructions can be applied verbatim and this substantially shortens our exposition. Essentially, we assume that H is a non-bipartite graph for which the problem GREEDY(partial ordering, undirected graphs, H-colourable) is not $\mathbf{\Sigma_2^p}$-complete and apply a sequence of constructions to yield that a known $\mathbf{\Sigma_2^p}$-complete problem is not complete, thereby obtaining a contradiction. Our 'known' $\mathbf{\Sigma_2^p}$-complete problem is GREEDY(partial ordering, undirected graphs, 3-colourable).

5.2 A complete problem

Theorem 5.1 *The problem GREEDY(partial ordering, undirected graphs, 3-colourable) is $\mathbf{\Sigma_2^p}$-complete.*

Proof Consider the problem GREEDY(partial ordering, undirected graphs, 3-colourable) (defined using the algorithm introduced in Chapter 2). Note that the basic property of a graph being 3-colourable is an **NP**-complete property (and not a polynomial time property like the ones considered before), therefore the complexity class in which this problem resides is $\mathbf{\Sigma_2^P}$. Throughout this proof, the problem GREEDY(partial ordering, undirected graphs, 3-colourable) shall be denoted $\mathcal{G}$. We shall prove completeness by reducing from the problem NOT CERTAIN 3-COLOURING OF BOOLEAN EDGE-LABELLED GRAPHS, henceforth to be abbreviated as problem $\mathcal{N}$. An instance of $\mathcal{N}$ of size n consists of an undirected graph O on n vertices, some of whose edges are labelled with the disjunction of two (possibly identical) literals over the set of Boolean variables $\{X_{i,j} : i, j = 1, 2, \ldots, n\}$ (the same literal may appear in more than one disjunction). A truth assignment t on the Boolean variables of $\{X_{i,j} : i, j = 1, 2, \ldots, n\}$ makes some of the labels on the edges of O true and some false. Form the graph $t(O)$ by retaining the edges labelled true, as well as any unlabelled edges, and dispensing with the edges labelled false. A yes-instance is an instance O for which there exists a truth assignment t resulting in a graph $t(O)$ that cannot be 3-coloured. This problem was proven to be $\mathbf{\Sigma_2^P}$-complete in [36].

Given an instance O of the problem $\mathcal{N}$, we shall construct an instance (G, P, s, x) of the problem $\mathcal{G}$ where G is an undirected graph, P is a partial ordering on these same vertices and s and x are two distinguished vertices. Moreover, O will be a yes-instance of $\mathcal{N}$ if, and only if, (G, P, s, x) is a yes-instance of $\mathcal{G}$; and the construction will be such that it can be completed using logspace.

Let $O = (U, F)$ and suppose that $U = \{1, 2, \ldots, n\}$. We build the undirected graph G from O as follows.

(a) For each vertex $i \in U$, 'attach' a copy of K_4 by identifying vertex i with one of the vertices of the clique. Denote the other three vertices by a_i, b_i^1 and b_i^2. We refer to the original vertices of U as *O-vertices*, the vertices of $\{a_i : i = 1, 2, \ldots, n\}$ as *a-vertices* and the vertices of $\{b_i^1, b_i^2 : i = 1, 2, \ldots, n\}$ as *b-vertices*.

(b) Retain any unlabelled edge (i, j) of F (between O-vertices i and j).

(c) For any labelled edge (i, j) of F (between O-vertices i and j), where $i < j$ and where the label is $L_{i,j}^1 \vee L_{i,j}^2$, replace the edge with a copy of the graph G_1 shown in Figure 5.1. We use, for example, $L_{i,j}^1$ to refer to the first literal labelling edge (i, j) and also a vertex within a graph G_1: this causes no confusion. The vertices of $\{L_{i,j}^1, L_{i,j}^2, \bar{L}_{i,j}^1, \bar{L}_{i,j}^2 : (i, j) \in F$, where $i < j\}$ are called *L-vertices*. Every L-vertex of any G_1 has an associated literal, *e.g.*, if the literal $L_{4,6}^1 = \neg X_{3,2}$ then the associated literal of vertex $L_{4,6}^1$ is $\neg X_{3,2}$ and the associated literal of vertex $\bar{L}_{4,6}^1$ is $X_{3,2}$. So, an L-vertex of some G_1 might have the same associated literal as an L-vertex of some other G_1. Finally, the vertices of $\{c_{i,j} : i, j = 1, 2, \ldots, n\}$ are called *c-vertices*, the vertices of $\{d_{i,j} : i, j = 1, 2, \ldots, n\}$ are called *d-vertices* and the vertices of $\{e_{i,j}^1, e_{i,j}^2 : i, j = 1, 2, \ldots, n\}$ are called *e-vertices*.

(d) Include a disjoint copy of K_4, whose vertices are $\{y, z, w, x\}$ and join vertices y, z and w to every a-vertex. Include the vertex s as an independent vertex.

Our partial ordering P is defined as follows. First, order the Boolean variables $\{X_{i,j} : i, j = 1, 2, \ldots, n\}$ lexicographically as

$$X_{1,1}, X_{1,2}, X_{1,3}, \ldots, X_{1,n}, X_{2,1}, X_{2,2}, \ldots, X_{n,n}$$

and denote this ordering by $<_X$; so $X_{1,1} <_X X_{1,2} <_X \ldots <_X X_{n,n}$. Next, consider the L-vertices. We obtain the notions of a positive L-vertex, where the vertex has an associated positive literal, and a negative L-vertex, where the vertex has an associated negative literal. Order the positive L-vertices so that if vertex λ_i is less than vertex λ_j in this ordering then the associated literal of λ_i is less than or equal to the associated literal of λ_j with respect to the ordering $<_X$ (note that there may be a number of such orderings on the positive L-vertices: it does not matter which of them we use). We obtain an analogous ordering of the negative L-vertices by taking complements (note that for every positive L-vertex $L^m_{i,j}$ or $\bar{L}^m_{i,j}$ with label l, the vertex $\bar{L}^m_{i,j}$ or $L^m_{i,j}$, respectively, is a negative L-vertex with label $\neg l$; and vice versa). As we walk down these two orderings in a synchronous fashion, the pairs of L-vertices are always complementary as are the pairs of associated literals. Denote these orderings as $\lambda_1 < \lambda_2 < \ldots < \lambda_k$ and $\mu_1 < \mu_2 < \ldots < \mu_k$, respectively, where $\{\lambda_i, \mu_i : i = 1, 2, \ldots, k\} = \{L^1_{i,j}, L^2_{i,j}, \bar{L}^1_{i,j}, \bar{L}^2_{i,j} : (i,j) \in F, \text{ where } i < j\}$.

Our partial ordering P begins as follows. The vertex s is less than both λ_1 and μ_1; and then we have the orderings $\lambda_1 < \lambda_2 < \ldots < \lambda_k$ and $\mu_1 < \mu_2 < \ldots < \mu_k$. Also, for any index $i \in \{1, 2, \ldots, k-1\}$, if the associated literal of λ_i is different from the associated literal of λ_{i+1} then additionally $\lambda_i < \mu_{i+1}$ and $\mu_i < \lambda_{i+1}$. In order to complete P, choose any linear ordering of the c-vertices, followed by any linear ordering of the d-vertices, followed

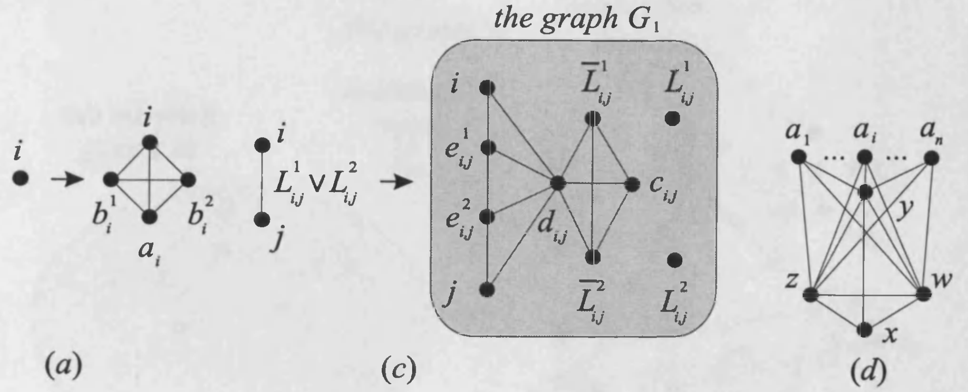


Figure 5.1: Phases (a), (c) and (d) of constructing G from O.

by any linear ordering of the e-vertices, followed by the ordering $1, 2, \ldots, n$ of the O-vertices, followed by any linear ordering of the b-vertices, followed by any linear ordering of the a-vertices, followed by the ordering w, y, z, x; and additionally define that both λ_k and μ_k are less than the least c-vertex (if there are no L-vertices then just concatenate the linear ordering of the c-vertices after the vertex s).

The construction of (G, P, s, x) from O is illustrated in Figure 5.2 (note that to avoid cluttering the figure, not all vertices are named; and that the bold edges correspond to the structure of O). Clearly, this construction can be completed using logspace.

Suppose that O is a yes-instance of problem $\mathcal{N}$. Hence, there exists a truth assignment t such that $t(O)$ is not 3-colourable. Consider the execution of the algorithm GREEDY(3-colourable) on instance (G, P, s) where the chosen linear ordering in P is that induced by the truth assignment t; that is, an

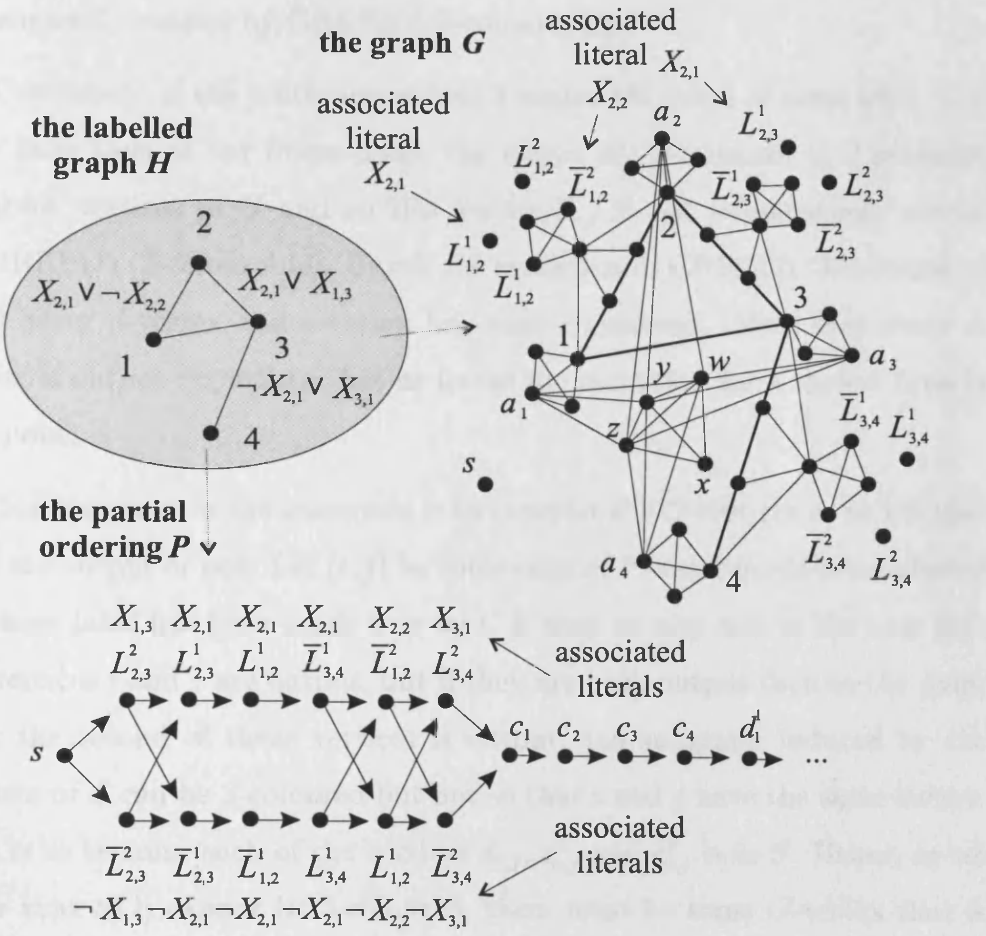


Figure 5.2: The construction of (G, P, s, x) from O.

L-vertex is chosen if, and only if, its associated Boolean literal is set at true by t. The first point to note is that s and every L-vertex chosen is output by GREEDY(3-colourable), as is every c-vertex. Let us freeze the execution at this point. Note that if the truth assignment t makes the label of some edge (i, j) of F true then at our freeze-point, the vertex $d_{i,j}$ is adjacent to at most 2 vertices of S (the set of chosen vertices), and so this vertex $d_{i,j}$ is subsequently output by GREEDY(3-colourable).

Conversely, if the truth assignment t makes the label of some edge (i, j) of F false then at our freeze-point, the vertex $d_{i,j}$ is adjacent to 3 mutually adjacent vertices of S and so this vertex $d_{i,j}$ is not subsequently output by GREEDY(3-colourable). Unroll the execution of GREEDY(3-colourable) until every d-vertex and e-vertex has been considered. Note that every e-vertex is output regardless. Let us freeze the execution for a second time at this point.

Our next task in the execution is to consider the O-vertices as to whether they are output or not. Let (i, j) be some edge of F which is either unlabelled or whose label has been made true by t. It may or may not be the case that the vertices i and j are output; but if they are both output then at the point after the second of these vertices is output, the subgraph induced by the vertices of S can be 3-coloured but not so that i and j have the same colour. This is so because each of the vertices $d_{i,j}$, $e^1_{i,j}$ and $e^2_{i,j}$ is in S. Hence, as we know that $t(O)$ cannot be 3-coloured, there must be some O-vertex that is not output; and, consequently, there is at least one a-vertex output. Having an a-vertex output means that not all of $\{y, z, w\}$ are output which in turn means that x is output. Hence, (G, P, s, x) is a yes-instance of problem $\mathcal{G}$.

Conversely, suppose that (G, P, s, x) is a yes-instance of problem $\mathcal{G}$. Fix an accepting execution of the algorithm GREEDY(3-colourable) on input (G, P, s) and denote the linear ordering chosen within P by τ. This execution gives rise to a truth assignment t on the literals labelling the edges of the graph O: if τ is such that a positive L-vertex, with associated literal $X_{i,j}$, say, is chosen then set $t(X_{i,j})$ to be true; and if τ is such that a negative L-vertex, with associated literal $\neg X_{i,j}$, say, is chosen then set $t(X_{i,j})$ to be false (note that this truth assignment is well-defined). As before, every L-vertex on τ is output by GREEDY(3-colourable); and, by arguing as we did earlier, for any $i, j \in \{1, 2, \dots, n\}$ with $i < j$ and where (i, j) is a labelled edge of O, the truth assignment t makes $L^1_{i,j} \vee L^2_{i,j}$ true if, and only if, the vertices $d_{i,j}$, $e^1_{i,j}$ and $e^2_{i,j}$ are output.

At various points in the execution of GREEDY(3-colourable), a check is made to see whether the vertices of S induce a 3-colourable graph. Consider such a check and suppose that the vertices of $\{d_{i,j}, e^1_{i,j}, e^2_{i,j}\}$ have been placed in S. Consider the subgraph K of G induced by those vertices that are both in S and in the copy of G_1 pertaining to the labelled edge (i, j) of O. In particular, consider the role of K when it comes to attempting to colour the subgraph of G induced by the vertices of S. A simple combinatorial verification yields that the role of the vertices of K is to allow i and j to be coloured with any pair of distinct colours but not with identical colours. Hence, any check to see whether the subgraph of G induced by the vertices of S can be 3-coloured is equivalent to a check of whether the subgraph of $t(O)$ induced by (vertices corresponding to) the O-vertices of S can be 3-coloured. We know that our accepting computation on (G, P, s, x) outputs x. This can only happen if not all of $\{y, z, w\}$ are output, *i.e.*, if at least one a-vertex, a_m, say, is output, *i.e.*, if the O-vertex m is not output, *i.e.*, if the graph $t(O)$ can not be 3-coloured. The result follows. □

5.3 The construction

We now prove our main result using the techniques originating with Hell and Nešetřil. Of course, these techniques have to be adapted to our scenario.

Theorem 5.2 *The problem GREEDY(partial ordering, undirected graphs, H-colourable) is* **NP***-complete, if H is bipartite, and* $\mathbf{\Sigma_2^p}$*-complete, if H is non-bipartite.*

Proof Throughout the proof we shall denote the problem GREEDY(partial ordering, undirected graphs, H-colourable) by $\mathcal{G}_H$. Clearly, $\mathcal{G}_H$ can be solved in $\mathbf{\Sigma_2^p}$, if H is non-bipartite, and in **NP**, if H is bipartite (the latter because the H-colourability problem, for H-bipartite, can be solved in polynomial time [23]). Moreover, because the property of being H-colourable, for H bipartite, is non-trivial on undirected graphs, hereditary, and polynomial time testable, by Corollary 3.15 we have that $\mathcal{G}_H$ is **NP**-complete if H is bipartite. Actually, note that if H is bipartite then $\mathcal{G}_H$ and the problem GREEDY(partial ordering, undirected graphs, bipartite) are one and the same.

To prove that for any non-bipartite graph H, the problem $\mathcal{G}_H$ is $\mathbf{\Sigma_2^p}$-complete, we will modify the proof of Theorem 1 of [23] which states that: '*If H is bipartite then the H-colouring problem is in* **P**. *If H is non-bipartite then the H-colouring problem is* **NP***-complete.*' The proof begins by detailing three ways of constructing a graph H' from a graph H such that if the H'-colouring problem is **NP**-complete then the H-colouring problem is **NP**-complete as well. We will show that such constructions can be used to prove

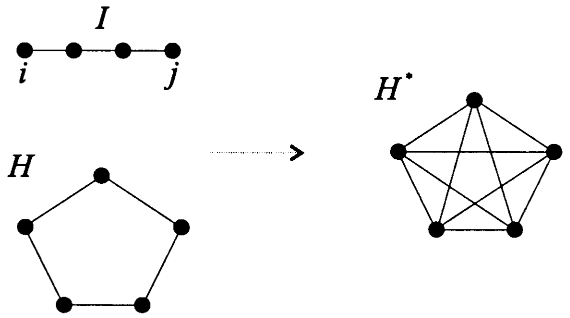


Figure 5.3: The indicator construction.

that the problem $\mathcal{G}_H$ is $\mathbf{\Sigma_2^p}$-complete.

Construction A: *The indicator construction.*

Let I be a fixed graph and let i and j be distinct vertices of I such that some automorphism of I maps i to j and j to i. The indicator construction (with respect to (I, i, j)) transforms a given graph H into a graph H^* defined to be the subgraph of H induced by all edges (h, h') for which there is a homomorphism of I to H mapping i to h and j to h'. Because of our assumptions on I, the edges of H^* will be undirected. The construction is illustrated in Figure 5.3. Note that we will always make sure that H^* does not contain any loops, *i.e.*, that no homomorphism of I to H can map i and j to the same vertex.

Lemma 5.3 *If the problem $\mathcal{G}_{H^*}$ is $\mathbf{\Sigma_2^p}$-complete then so is $\mathcal{G}_H$.*

Proof Assume that $\mathcal{G}_{H^*}$ is $\mathbf{\Sigma_2^p}$-complete; and so, in particular, H^* has at least one edge (otherwise H^* would be the empty graph and $\mathcal{G}_{H^*}$ would not be $\mathbf{\Sigma_2^p}$-complete). We will reduce $\mathcal{G}_{H^*}$ to $\mathcal{G}_H$ (via a logspace reduction). Let

(G^*, P^*, s^*, x^*) be an instance of $\mathcal{G}_{H^*}$. From it, we shall construct an instance (G, P, s, x) of $\mathcal{G}_H$.

Graph G is obtained from G^* as follows. For any vertex i of G^*, there is a corresponding vertex i of G: we will refer to such vertices of G as *G^*-vertices* (note how we consider the G^*-vertices of G and the vertices of G^* as being identically named). For any edge (u, v) of G^*, we add a copy of graph I to G by identifying the G^*-vertex u with vertex i in I and the G^*-vertex v with vertex j in I (all added copies of I are disjoint).

The partial ordering P consists of a linear ordering L (any one will do) on the vertices of G which are not G^*-vertices, and we concatenate on to this linear ordering the partial ordering P^* (of the G^*-vertices). Vertex s is the first vertex of the linear ordering L and vertex x is the G^*-vertex x^*. An illustration of this construction is depicted in Figure 5.4 (where the graphs I, H and H^* are as in Figure 5.3).

Consider the algorithm GREEDY(H-colourable) on the input (G, P, s). As H^* contains at least one edge, there is a homomorphism from I to H. Hence, as the linear ordering L consists of disjoint copies of $I \setminus \{i, j\}$, GREEDY(H-colourable) outputs every vertex of L. After consideration of the vertices of L, GREEDY(H-colourable) is working with essentially the same partial ordering as is the algorithm GREEDY(H^*-colourable) initially on input (G^*, P^*, s^*); so consider executions of these algorithms with respect to the same subsequent linear ordering.

Our induction hypothesis is as follows: 'The current-vertex in both executions is s_0; GREEDY(H-colourable) has so far output the vertices of

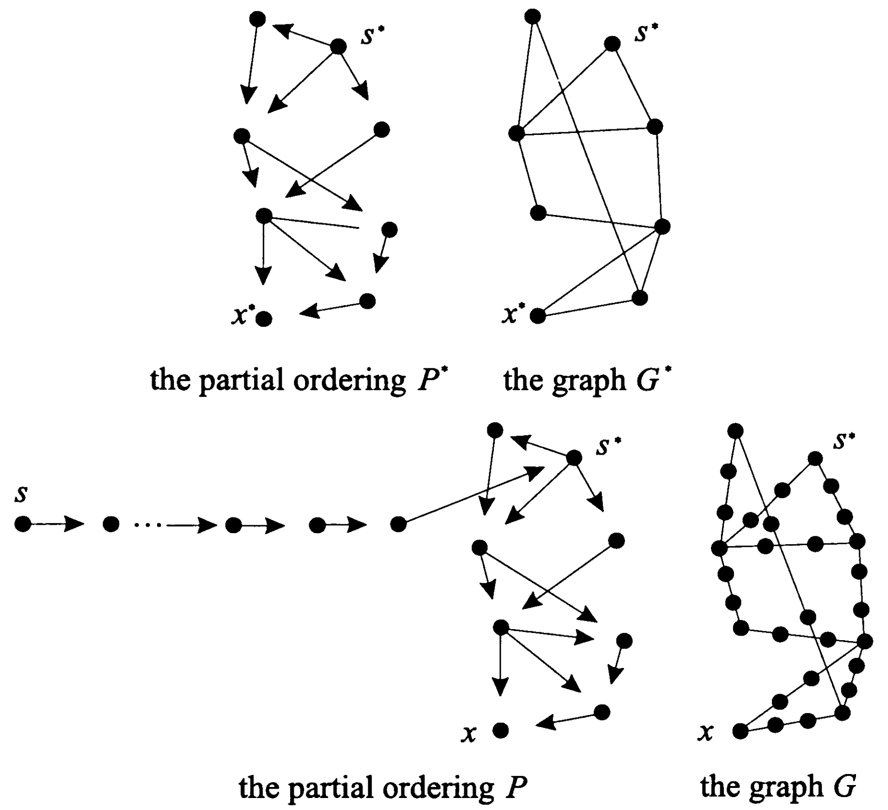


Figure 5.4: Building (G, P, s, x) from (G^*, P^*, s^*, x^*).

$L \cup \{s_1, s_2, \ldots, s_m\}$, where vertex s_i is a G^*-vertex, for $i = 1, 2, \ldots, m$; and GREEDY(H^*-colourable) has so far output the vertices of $\{s_1, s_2, \ldots, s_m\}$.'

Suppose that the induction hypothesis holds at some point (it certainly holds when $s_0 = s^*$).

Suppose that GREEDY(H^*-colouring) outputs the vertex s_0. This means that there exists an homomorphism $f^* : \langle\{s_0, s_1, \ldots, s_m\}\rangle_{G^*} \to H^*$. By construction of H^*, there must exist a homomorphism $f : \langle L \cup \{s_0, s_1, \ldots, s_m\}\rangle_G \to H$, where $f(s_i) = f^*(s_i)$, for $i = 0, 1, \ldots, m$, and $f(v)$ is the 'natural' map for $v \in L$ (derived from the definition of H^* from H). Hence, GREEDY(H-colourable) outputs the vertex s_0.

Conversely, suppose that GREEDY(H-colourable) outputs the vertex s_0. This means that there exists a homomorphism $f : \langle L \cup \{s_0, s_1, \ldots, s_m\}\rangle_G \to H$. Again by construction of H^*, there must exist a homomorphism $f^* : \langle\{s_0, s_1, \ldots, s_m\}\rangle_{G^*} \to H_*$, where $f^*(s_i) = f(s_i)$, for $i = 0, 1, \ldots, m$. Hence, GREEDY(H^*-colouring) outputs the vertex s_0. The result follows by induction. $\square$

Construction B: *The sub-indicator construction.*

Let J be a fixed graph with specified (distinct) vertices $j, k_1, k_2, \ldots, k_t$, for some $t \geq 1$. The sub-indicator construction (with respect to $J, j, k_1, k_2, \ldots, k_t$) transforms a given graph H with t (distinct) specified vertices $h_1, h_2, \ldots, h_t$ to its subgraph $\tilde{H}$ induced by the vertex set $\tilde{V}$ defined as follows. A vertex v of H belongs to $\tilde{V}$ just if there exists a homomorphism of J to H taking k_i to h_i, for $i = 1, 2, \ldots, t$, and taking j to v. An illustration of this construction is depicted in Figure 5.5 (where, for clarity, we have shown the vertices of H excluded from $\tilde{H}$).

Lemma 5.4 *If the problem $\mathcal{G}_{\tilde{H}}$ is $\mathbf{\Sigma_2^p}$-complete then so is $\mathcal{G}_H$.*

Proof Assume that $\mathcal{G}_{\tilde{H}}$ is $\mathbf{\Sigma_2^p}$-complete; and so, in particular, $\tilde{H}$ has at least one vertex. We will reduce $\mathcal{G}_{\tilde{H}}$ to $\mathcal{G}_H$ (via a logspace reduction). Let $(\tilde{G}, \tilde{P}, \tilde{s}, \tilde{x})$ be an instance of $\mathcal{G}_{\tilde{H}}$. From it, we shall construct an instance (G, P, s, x) of $\mathcal{G}_H$.

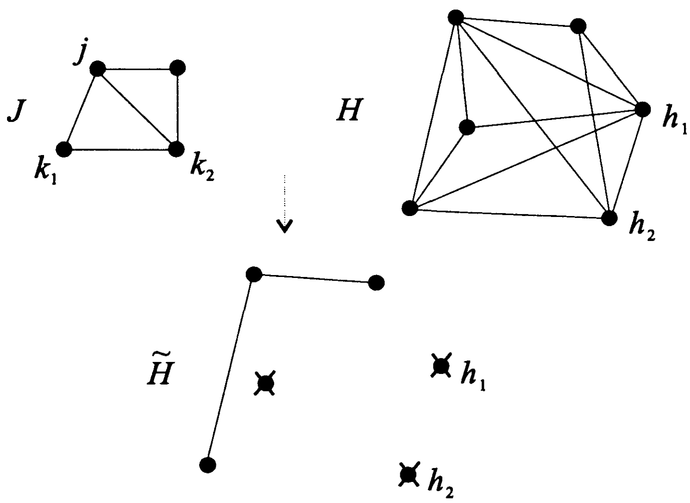


Figure 5.5: Building $\tilde{H}$ from H and J.

The graph G is built from: a copy of $\tilde{G}$, of size n; a copy of H; and n copies of J (with J and H prior to the statement of the lemma), by identifying the vertex k_i in any copy of J with the vertex h_i of H, for $i = 1, 2, \ldots, t$, and identifying the vertex j in the i^{th} copy of J with the i^{th} vertex of $\tilde{G}$, for $i = 1, 2, \ldots, n$. The vertices of G corresponding to the vertices of $\tilde{G}$ (and the vertices j of the copies of J) are called *$\tilde{G}$-vertices*, the vertices of G corresponding to the vertices of the copies of J but different from $j, k_1, k_2, \ldots, k_t$ are called *J-vertices*, and the vertices of G corresponding to the vertices of H are called *H-vertices*.

The partial ordering P consists of any linear ordering of the H-vertices, concatenated onto any linear ordering of the J-vertices concatenated onto the ordering $\tilde{P}$ of the $\tilde{G}$-vertices. The vertex s is the first H-vertex in the ordering P and the vertex x is the vertex $\tilde{x}$ of $\tilde{P}$. The whole construction

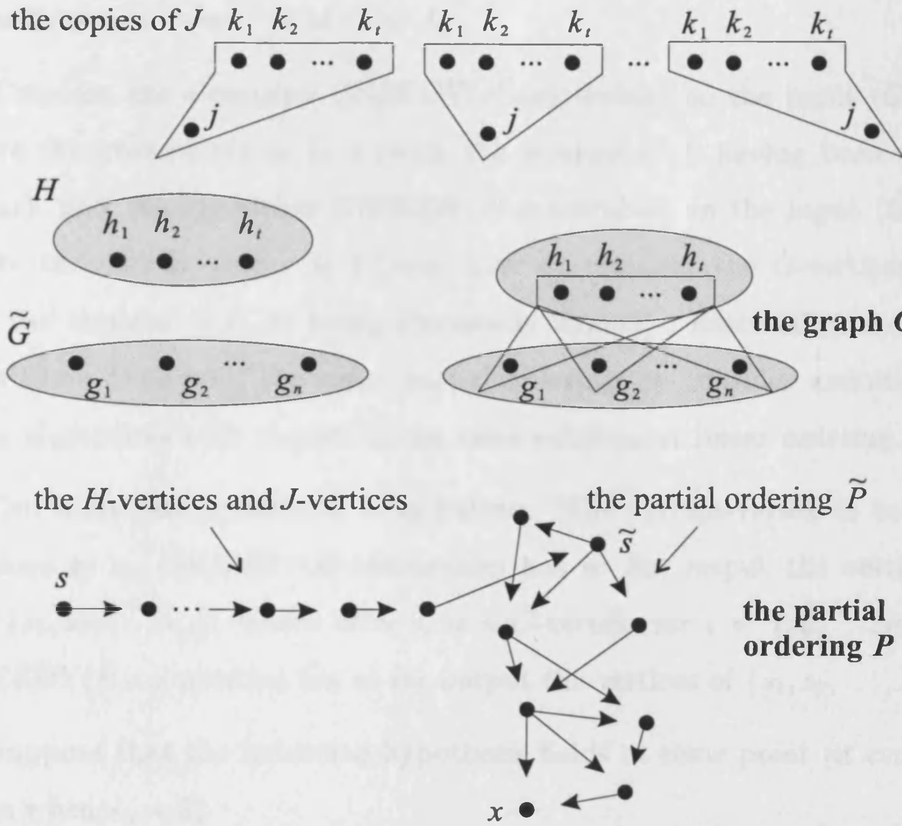


Figure 5.6: Building G from H, copies of J and $\tilde{G}$.

can be pictured in Figure 5.6. Clearly, this construction can be undertaken using logspace.

We begin by showing that any execution of GREEDY(H-colourable) on input (G, P, s) outputs every H-vertex and J-vertex of G. Clearly every H-vertex is output. Consider some copy of J (used in the formation of G). As $\tilde{H}$ has at least one vertex, there is a homomorphism from J to H taking k_i to h_i, for $i = 1, 2, \ldots, t$. Hence, every J-vertex is output. Denote the set of

H-vertices and J-vertices of G by L.

Consider the algorithm GREEDY(H-colourable) on the input (G, P, s), where the current-vertex is $\tilde{s}$ (with the vertices of L having been output so far), and the algorithm GREEDY($\tilde{H}$-colourable) on the input $(\tilde{G}, \tilde{P}, \tilde{s})$ where the current-vertex is $\tilde{s}$ (note how we consider the $\tilde{G}$-vertices of G and the vertices of $\tilde{G}$ as being identically named). Essentially, these two algorithms work with the same partial ordering; so consider executions of these algorithms with respect to the same subsequent linear ordering.

Our induction hypothesis is as follows: 'The current-vertex in both executions is s_0; GREEDY(H-colourable) has so far output the vertices of $L \cup \{s_1, s_2, \ldots, s_m\}$, where each s_i is a $\tilde{G}$-vertex, for $i = 1, 2, \ldots, m$; and GREEDY($\tilde{H}$-colourable) has so far output the vertices of $\{s_1, s_2, \ldots, s_m\}$.'

Suppose that the induction hypothesis holds at some point (it certainly holds when $s_0 = \tilde{s}$).

Suppose that s_0 is output by GREEDY(H-colourable). That is, there is a homomorphism $f : \langle L \cup \{s_0, s_1, \ldots, s_m\}\rangle_G \to H$. In particular: $f(s_i)$ is a vertex of $\tilde{H}$, for $i = 0, 1, \ldots, m$; and if (s_i, s_j) is an edge of $\tilde{G}$ then $(f(s_i), f(s_j))$ is an edge of $\tilde{H}$, for $i, j = 0, 1, \ldots, m$. Hence, we have a homomorphism $\tilde{f} : \langle\{s_0, s_1, \ldots, s_m\}\rangle_{\tilde{G}} \to \tilde{H}$, and so s_0 is output by GREEDY($\tilde{H}$-colourable).

Conversely, suppose that s_0 is output by GREEDY($\tilde{H}$-colourable). That is, there is a homomorphism $\tilde{f} : \langle\{s_0, s_1, \ldots, s_m\}\rangle_{\tilde{G}} \to \tilde{H}$. Consider the copy of J corresponding to the $\tilde{G}$-vertex s_i of G. As $\tilde{f}(s_i)$ is a vertex of $\tilde{H}$, $\tilde{f}$ can be extended to a homomorphism $f : \langle L \cup \{s_0, s_1, \ldots, s_m\}\rangle_G \to H$. Hence, s_0 is output by GREEDY(H-colourable). The result follows by induction. $\square$

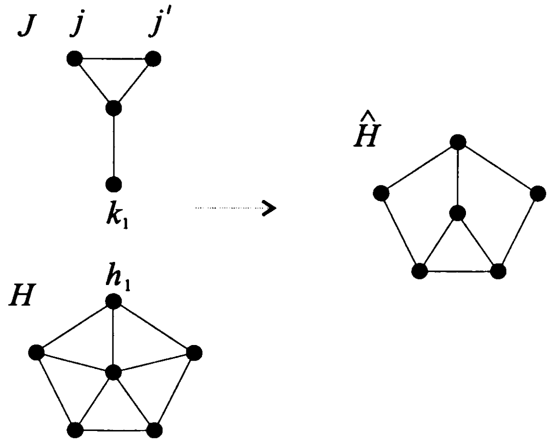


Figure 5.7: Building $\hat{H}$ from H and J.

Construction C: The edge-sub-indicator construction.

Let J be a fixed graph with a specified edge (j, j') and t specified vertices $k_1, k_2, \ldots, k_t$, such that all vertices $j, j', k_1, k_2, \ldots, k_t$ are distinct and some automorphism of J keeps $k_1, k_2, \ldots, k_t$ fixed while exchanging the vertices j and j'. The edge-sub-indicator construction transforms a given graph H with t (distinct) specified vertices $h_1, h_2, \ldots, h_t$ into its subgraph $\hat{H}$ induced by those edges (h, h') of H for which there is a homomorphism of J to H taking k_i to h_i, for $i = 1, 2, \ldots, t$, and j to h and j' to h'. The construction can be visualised as in Figure 5.7.

Lemma 5.5 *If the problem $\mathcal{G}_{\hat{H}}$ is $\mathbf{\Sigma_2^P}$-complete then so is $\mathcal{G}_H$.*

Proof Assume that $\mathcal{G}_{\hat{H}}$ is $\mathbf{\Sigma_2^P}$-complete; and so, in particular, $\hat{H}$ has at least one edge. We will reduce $\mathcal{G}_{\hat{H}}$ to $\mathcal{G}_H$ (via a logspace reduction). Let $(\hat{G}, \hat{P}, \hat{s}, \hat{x})$ be an instance of $\mathcal{G}_{\hat{H}}$. From it, we shall construct an instance

(G, P, s, x) of $\mathcal{G}_H$.

The graph G is constructed from: a copy of $\hat{G}$, with e edges; a copy of H; and e copies of J (with H and J as prior to the statement of this lemma), by identifying every vertex k_i in any copy of J with the vertex h_i of H, for $i = 1, 2, \ldots, t$, and each edge e of $\hat{G}$ with the edge (j, j') of a unique copy of J. The vertices of G corresponding to the vertices of $\hat{G}$ (and the vertices j and j' of the copies of J) are called *$\hat{G}$-vertices*, the vertices of G corresponding to the vertices of the copies of J but different from $j, k_1, k_2, \ldots, k_t$ are called *J-vertices*, and the vertices of G corresponding to the vertices of H are called *H-vertices*.

The partial ordering P consists of any linear ordering of the H-vertices, concatenated onto any linear ordering of the J-vertices concatenated onto the ordering $\hat{P}$ of the $\hat{G}$-vertices. The vertex s is the first H-vertex in the ordering P and the vertex x is the vertex $\hat{x}$ of $\hat{P}$. The whole construction can be pictured in Figure 5.8. Clearly, this construction can be undertaken using logspace.

We begin by showing that any execution of GREEDY(H-colourable) on input (G, P, s) outputs every H-vertex and J-vertex of G. Clearly every H-vertex is output. Consider some copy of J (used in the formation of G). As $\hat{H}$ has at least one edge, there is a homomorphism from J to H taking k_i to h_i, for $i = 1, 2, \ldots, t$. Hence, every J-vertex is output. Denote the set of H-vertices and J-vertices of G by L.

Consider the algorithm GREEDY(H-colourable) on the input (G, P, s), where the current-vertex is $\hat{s}$ (with the vertices of L having been output

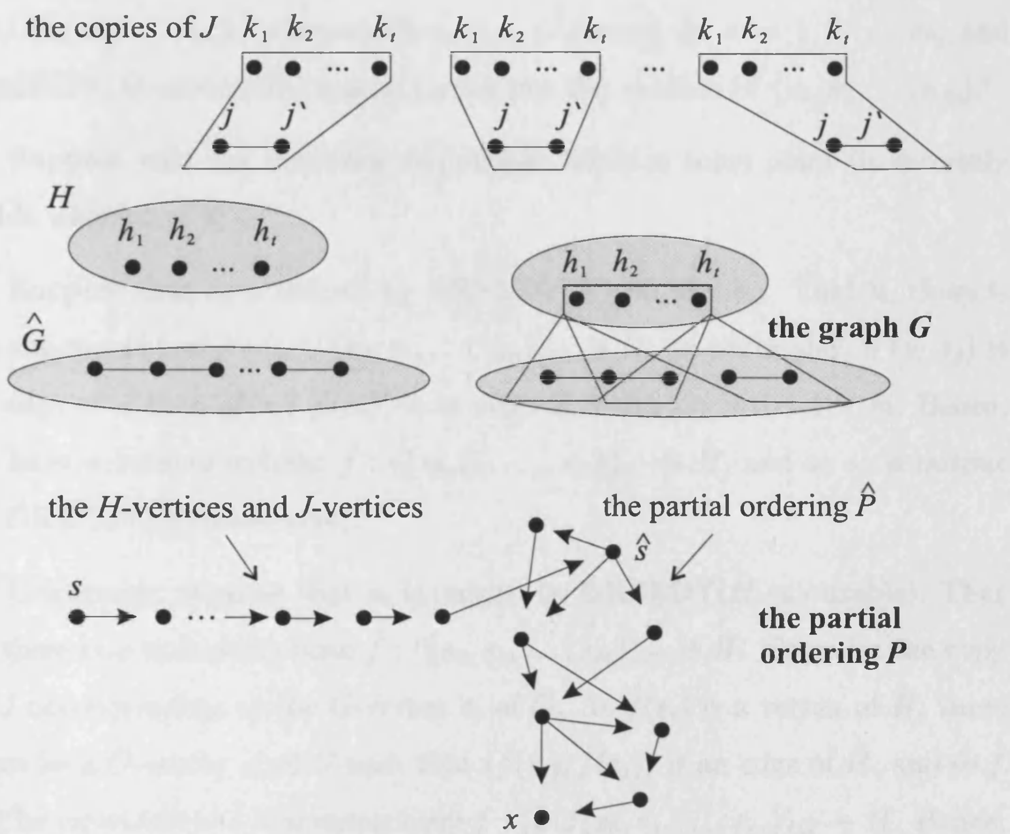


Figure 5.8: Building G from H, copies of J and $\hat{G}$.

so far), and the algorithm GREEDY($\hat{H}$-colourable) on the input $(\hat{G}, \hat{P}, \hat{s})$ where the current-vertex is $\hat{s}$ (note how we consider the $\hat{G}$-vertices of G and the vertices of $\tilde{G}$ as being identically named). Essentially, these two algorithms work with the same partial ordering; so consider executions of these algorithms with respect to the same subsequent linear ordering.

Our induction hypothesis is as follows: 'The current-vertex in both executions is s_0; GREEDY(H-colourable) has so far output the vertices of

$L \cup \{s_1, s_2, \ldots, s_m\}$, where each s_i is a $\hat{G}$-vertex, for $i = 1, 2, \ldots, m$; and GREEDY($\hat{H}$-colourable) has so far output the vertices of $\{s_1, s_2, \ldots, s_m\}$.'

Suppose that the induction hypothesis holds at some point (it certainly holds when $s_0 = \hat{s}$).

Suppose that s_0 is output by GREEDY(H-colourable). That is, there is a homomorphism $f : \langle L \cup \{s_0, s_1, \ldots, s_m\}\rangle_G \to H$. In particular, if (s_i, s_j) is an edge of $\hat{G}$ then $(f(s_i), f(s_j))$ is an edge of $\hat{H}$, for $i, j = 0, 1, \ldots, m$. Hence, we have a homomorphism $\hat{f} : \langle\{s_0, s_1, \ldots, s_m\}\rangle_{\hat{G}} \to \hat{H}$, and so s_0 is output by GREEDY($\hat{H}$-colourable).

Conversely, suppose that s_0 is output by GREEDY($\hat{H}$-colourable). That is, there is a homomorphism $\hat{f} : \langle\{s_0, s_1, \ldots, s_m\}\rangle_{\hat{G}} \to \hat{H}$. Consider the copy of J corresponding to the $\hat{G}$-vertex s_i of G. As $\hat{f}(s_i)$ is a vertex of $\hat{H}$, there must be a $\hat{G}$-vertex s_j of G such that $(\hat{f}(s_i), \hat{f}(s_j))$ is an edge of $\hat{H}$, and so $\hat{f}$ can be extended to a homomorphism $f : \langle L \cup \{s_0, s_1, \ldots, s_m\}\rangle_G \to H$. Hence, s_0 is output by GREEDY(H-colourable). The result follows by induction. □

Now we can proceed as Hell and Nešetřil did in [23]. Assume that there exists a non-bipartite graph H for which the problem $\mathcal{G}_H$ is not $\mathbf{\Sigma_2^p}$-complete. Choose H so that it is non-bipartite and the problem $\mathcal{G}_{H'}$ is $\mathbf{\Sigma_2^p}$-complete for any non-bipartite graph H':

(i) with fewer vertices than H; or

(ii) with the same number of vertices as H but with more edges.

It is straightforward to see that, under the assumption above, such an H must exist.

In [23], working from a similar hypothesis and graph H, the proof proceeds by using the indicator, sub-indicator and edge-sub-indicator constructions, in tandem with lemmas analogous to Lemmas 5.3, 5.4 and 5.5, to show that H must be a 3-clique; and hence that the 3-colouring problem is not **NP**-complete, thus yielding a contradiction. The sections of the proof of the main theorem of [23] entitled 'The structure of triangles' and 'The structure of squares' can be applied verbatim to our graph H (as the constructions we use are identical and we have our analogous Lemmas 5.3, 5.4 and 5.5). Hence, we may assume that H is 3-colourable, *i.e.*, that H is a 3-clique. However, Theorem 5.1 yields a contradiction as the problem GREEDY(partial ordering, undirected graphs, H-colourable) is none other than $\mathcal{G}_H$ when H is a 3-clique, and the result follows. □

5.4 Conclusion

In this chapter, we have exhibited a complexity-theoretic dichotomy result concerning the nondeterministic computation of lexicographically first maximal H-colourable subgraphs of graphs. Our dichotomy result is different from other dichotomy results in that it is concerned with **NP**-completeness and $\mathbf{\Sigma_2^p}$-completeness, as opposed to computability in polynomial time and **NP**-completeness as is more often the case. There are natural directions in which to extend this research.

> *Can we obtain a constructive proof (as opposed to the proof by contradiction above) of our main result?*

Can we obtain a similar result in the case of directed graphs or other structures?

What is the complexity of the analogously defined lexicographically last maximal subgraph problem (again, with respect to an appropriate property π), in the cases when a graph is linearly ordered and partially ordered?

The only result we know of as regards computing lexicographically last subgraphs is that of [25] where it is proven that deciding whether a given set of vertices of a given linearly ordered graph is the lexicographically last such maximal independent set is **co-NP**-complete. Regarding the first two questions, it is open as to whether there is a constructive proof of Hell and Nešetřil's result and also whether it can be extended to directed graphs; and it therefore not surprising that so far we have not been able to answer these questions. We have therefore decided to extend our research to try and obtain another dichotomy result when our partial ordering is replaced with a linear ordering (so that the two relevant complexity classifications are 'computable in polynomial time' and 'computable in $\mathbf{P}^{\mathbf{NP}}$'). We will present our results in the next chapter.

Chapter 6

Linear orderings

6.1 Introduction

In the previous chapter we presented a dichotomy result involving the problem GREEDY(partial ordering, undirected graphs, H-colourable): we proved that the problem is **NP**-complete if H is bipartite, and $\mathbf{\Sigma_2^p}$-complete otherwise. In this chapter we will consider the analogous problem except when the vertices of any graph are linearly, as opposed to partially, ordered. We will therefore return to the world of lexicographically first maximal subgraph problems considered by Miyano in [29]. Miyano [30] also proved that if a property π is hereditary, determined by the blocks, non-trivial on connected graphs and testable in polynomial time then the problem of deciding whether a given vertex of a given undirected graph G, whose vertices are linearly ordered, lies in the lexicographically first maximal connected subgraph of G satisfying π is $\mathbf{\Delta_2^p}$-complete. His results do not apply to our problem because we consider properties testable in nondeterministic polynomial time, and because we do not require the subgraph induced by the set of vertices output by our algorithm, on a specific instance, to be connected.

We will prove here another dichotomy result; that is, that the problem GREEDY(linear ordering, undirected graphs, H-colourable) is **P**-complete if H is a bipartite graph, and $\mathbf{\Delta_2^p}$-complete otherwise. Following the strategy used in Chapter 5, we will first show that a particular problem is complete for $\mathbf{\Delta_2^p}$, and then use such a result to derive a contradiction and so obtain our main result.

6.2 Deterministic Satisfiability

In order to prove the completeness of the problem GREEDY(linear ordering, undirected graphs, 3-colourable) for $\mathbf{\Delta_2^p}$, we will reduce from the problem Deterministic Satisfiability, that was proved complete for $\mathbf{\Delta_2^p}$ by Papadimitriou in [32].

Definition 1. Let $Y_1, \ldots, Y_k, \{x_1, \ldots, x_{k-1}\}$ be disjoint sets of Boolean variables. A Boolean formula F in conjunctive normal form involving these variables is said to be *deterministic* if it consists of the conjunction of the following clauses.

- For each $y \in Y_1 \cup Y_k$ either (y) or $(\neg y)$ is a clause of F.
- For each $j = 1, \ldots, k-1$ and each variable $y \in Y_{j+1}$, there are two sets of conjunctions of literals over $Y_j \cup \{x_j\}$, called, respectively, C_y and D_y,

such that for each conjunction $\alpha \in C_y$ and each conjunction $\beta \in D_y$, $(\alpha \to y)$ and $(\beta \to \neg y)$ are both clauses of F; and, furthermore, for any truth assignment on $Y_j \cup \{x_j\}$, *exactly one* of the conjunctions in $C_y \cup D_y$ is true (note that F can be written in conjunctive normal form because α and β are conjunctions of literals, and therefore $(\alpha \to y)$ and $(\beta \to \neg y)$ can be written as disjunctions of literals).

We will now show a very simple example of a deterministic formula F, where $k = 3$, $Y_1 = \{y_1\}$, $Y_2 = \{y_2\}$ and $Y_3 = \{y_3\}$.

$$(y_1) \wedge (\neg y_3)$$

$$\wedge((y_1 \wedge x_1) \to y_2) \wedge((\neg y_1 \wedge x_1) \to y_2) \wedge((\neg y_1 \wedge \neg x_1) \to \neg y_2) \wedge((y_1 \wedge \neg x_1) \to \neg y_2)$$

$$\wedge((y_2 \wedge x_2) \to y_3) \wedge((\neg y_2 \wedge x_2) \to y_3) \wedge((\neg y_2 \wedge \neg x_2) \to \neg y_3) \wedge((y_2 \wedge \neg x_2) \to \neg y_3)$$

The first row corresponds to the first part of the definition, that is, for every variable y in $Y_1 \cup Y_3$ there is a clause in F. Because every satisfying truth assignment on the variables of F must satisfy these clauses, the values of y_1 and y_3 are effectively fixed. The second and third rows of the example correspond to the second part of the definition of a deterministic formula (note that we have written the clauses in the form of implications only for clarity). The sets of conjunctions are as follows: $C_{y_2} = \{(y_1 \wedge x_1), (\neg y_1 \wedge x_1)\}$, $D_{y_2} = \{(\neg y_1 \wedge \neg x_1), (y_1 \wedge \neg x_1)\}$, $C_{y_3} = \{(y_2 \wedge x_2), (\neg y_2 \wedge x_2)\}$ and $D_{y_3} = \{(\neg y_2 \wedge \neg x_2), (y_2 \wedge \neg x_2)\}$. For each truth assignment on $Y_1 \cup \{x_1\}$, exactly one of the conjunctions in $C_{y_2} \cup D_{y_2}$ evaluates to true. Therefore, as the value of y_1 is effectively fixed, assigning a value to variable x_1 fixes the values of the variables in Y_2, that is, the variable y_2. Variable y_2 must have value true if one of the conjunctions in C_{y_2} evaluates to true, and y_2 must have value false if one of the conjunctions in D_{y_2} is true.

Once a truth value has been given to all variables in $Y_1 \cup \{x_1\}$, effectively all the variables in Y_2 have been given a fixed value. The process can therefore be repeated with variable x_2. If assigning a value to x_2 makes one of the conjunctions in C_{y_3} true then y_3 must evaluate to true, and if one of the conjunctions in D_{y_3} evaluates to true then y_3 must be assigned the value false. After x_2 has been given a value, and effectively the value of y_3 has been fixed, then such a value of y_3 must satisfy the clause consisting of a literal from Y_3, that is, the clause $(\neg y_3)$.

Having given the definition of a deterministic formula, we can now define the problem.

Definition 2. *Deterministic Satisfiability* is defined as follows.
Instance: $F_0(x_1, \ldots, x_{k-1}, Y_1, \ldots, Y_k)$, $F_1(Y_1, Z_1), \ldots, F_{k-1}(Y_{k-1}, Z_{k-1})$ where: $\{x_1, \ldots, x_{k-1}\}, Y_1, \ldots, Y_k, Z_1, \ldots, Z_{k-1}$ are disjoint sets of Boolean variables; F_0 is a deterministic formula; and $F_1, \ldots, F_{k-1}$ are Boolean formulae in 3-conjunctive normal form.
Question: Is there a truth assignment $\tau(x_1, \ldots, x_{k-1}, Y_1, \ldots, Y_k)$ such that:

- F_0 is satisfied by τ, and
- $F_j(\tau(Y_j), Z_j)$ is satisfiable if, and only if, $\tau(x_j) =$ true.

Deterministic Satisfiability can be solved in $\mathbf{\Delta_2^p}$ by using the following algorithm. For every variable $y \in Y_1$, there is either a clause (y) or a clause $(\neg y)$ in F_0. Let $\tau(Y_1)$ denote a truth assignment on Y_1 that satisfies such clauses. Clearly any satisfying truth assignment on F_0 must agree with $\tau(Y_1)$. Using an oracle for satisfiability, check whether the formula $F_1(\tau(Y_1), Z_1)$ is satisfiable. If it is then set the value of x_1 to true, else set it to false. The obtained truth assignment on $Y_1 \cup \{x_1\}$ fixes the value of the variables in Y_2 (because of the structure of F_0); therefore this gives us a well defined truth assignment $\tau(Y_2)$ on the variables in Y_2. The algorithm can now check, using an oracle for satisfiability, whether $F_2(\tau(Y_2), Z_2)$ is satisfiable. If it is then set the value of x_2 to be true, else set it to be false. The obtained truth assignment on $Y_2 \cup \{x_2\}$ gives us a well defined truth assignment on Y_3. The execution continues as described above until the unique truth value on x_{k-1} has been obtained. This, together with $\tau(Y_{k-1})$ (the truth assignment on the variables in Y_{k-1} obtained using the method outlined before) uniquely gives us a truth assignment $\tau(Y_k)$ on the variables in Y_k. For every variable $y \in Y_k$, there is either a clause (y) or a clause $(\neg y)$ in F_0. If $\tau(Y_k)$ satisfies all such clauses from F_0 then the given instance is a yes-instance of the problem. It is straightforward to notice that if a truth assignment cannot be found using such a procedure then the given instance of Deterministic Satisfiability is a no-instance.

6.3 The complete problem

Theorem 6.1 *The problem GREEDY(linear ordering, undirected graphs, 3-colourable) is $\mathbf{\Delta_2^p}$-complete.*

Proof The problem (as defined in Chapter 2) is clearly in $\mathbf{\Delta_2^p}$ as it can be solved in polynomial time by a deterministic Turing machine that has access to an oracle for 3-colourability. To prove the completeness of GREEDY(linear ordering, undirected graphs, 3-colourable), we will reduce from the problem Deterministic Satisfiability (DSAT). Given an instance $(F_0, \ldots, F_{k-1})$ of DSAT we will construct an instance (G, P, u, v) of GREEDY(linear ordering, undirected graphs, 3-colourable), where G is an undirected graph, P is a linear ordering on the vertices of G, u is the first vertex in the linear ordering and v is a vertex of G. The instance (G, P, u, v) will be a yes-instance of GREEDY(linear ordering, undirected graphs, 3-colourable) if, and only if, $(F_0, \ldots, F_{k-1})$ is a yes-instance of DSAT, and the construction will be such that it can be completed using logspace.

Let $Y_i = \{y_i^1, \ldots, y_i^{d_i}\}$, for $i = 1, \ldots, k$, and let $Z_i = \{z_i^1, \ldots, z_i^{m_i}\}$, for $i = 1, \ldots, k-1$. We will divide the construction of G from $(F_0, \ldots, F_{k-1})$ into several phases.

Phase 1 We construct a triangle, whose vertices are labelled u, t, f. For each variable e in $\{x_1, \ldots, x_{k-1}\} \cup Y_1 \cup \ldots \cup Y_k \cup Z_1 \cup \ldots \cup Z_{k-1}$, we add to the graph 2 vertices labelled, respectively, e and $\neg e$, where $\neg e$ is the negation of variable e. We will refer to such vertices as *literal-vertices*, and to the set containing all literal-vertices as L. Each vertex e is adjacent to the corresponding vertex $\neg e$, and both are adjacent to vertex u. See Figure 6.1 for an illustration.

Phase 2 For each clause in F_0 consisting of a literal from Y_1, y_1^i, say, we add to the graph a triangle whose vertices are labelled $y_{1,0}^i, y_{1,1}^i, y_{1,2}^i$. We then join $y_{1,1}^i$ and $y_{1,2}^i$ to literal-vertex y_1^i, and join vertex $y_{1,0}^i$ to vertex f (note that if the literal was, say, $\neg y_1^i$ then we would add to the graph three

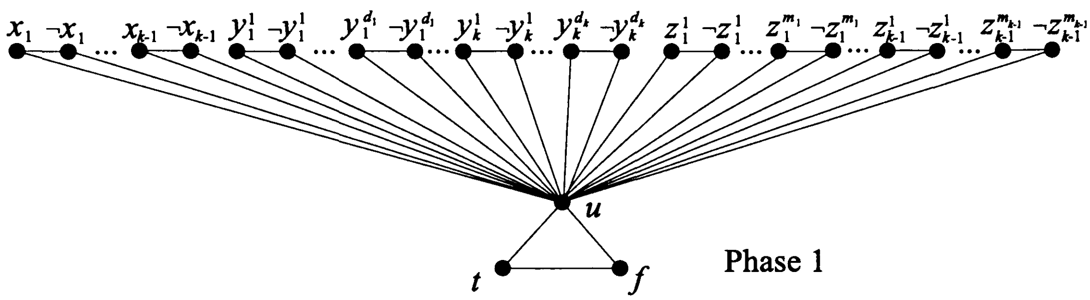


Figure 6.1: Phase 1 of the construction.

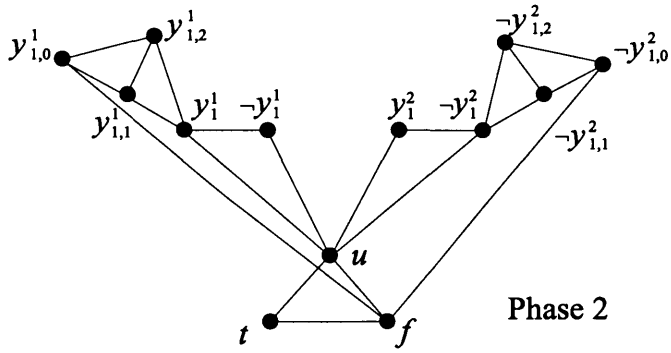


Figure 6.2: Phase 2 of the construction.

vertices labelled, respectively, $\neg y^i_{1,0}, \neg y^i_{1,1}, \neg y^i_{1,2}$, and we would join them as explained above). Notice that, by construction, in every proper 3-colouring of the graph, vertices $t, y^i_{1,0}$ and literal-vertex y^i_1 must be assigned the same colour. We will refer to the set of vertices added in this phase as V_1. See Figure 6.2 for an example relative to clauses (y^1_1) and $(\neg y^2_1)$.

Phase 3 The gadgets used in this phase are similar to the ones seen in Section 11.4.5 of [28]. For each formula F_i, where $i = 1, 2, \ldots, k-1$, we proceed as follows. For each clause j in F_i, we add to the graph 3 vertices labelled,

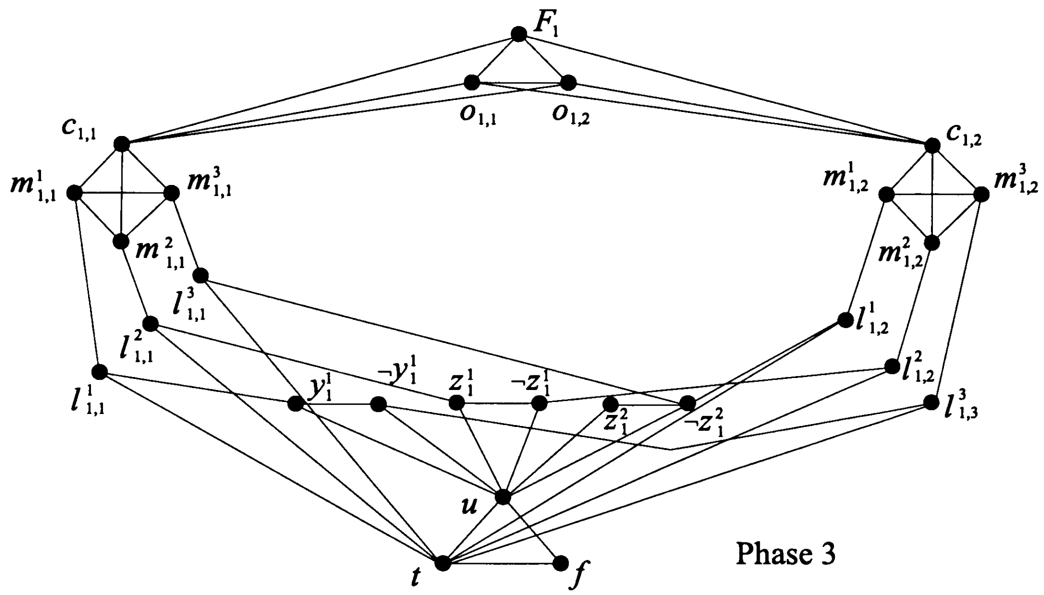


Figure 6.3: Phase 3 of the construction.

respectively, $l^1_{i,j}, l^2_{i,j}, l^3_{i,j}$. Such vertices correspond to the literals in the clause. Each of these vertices is adjacent to its corresponding literal-vertex, and is also adjacent to t. For each clause j in formula F_i, we then add to the graph a copy of a K_4, whose vertices are labelled $c_{i,j}, m^1_{i,j}, m^2_{i,j}$ and $m^3_{i,j}$ (all such copies are disjoint). We then join vertex $m^1_{i,j}$ to $l^1_{i,j}$, $m^2_{i,j}$ to $l^2_{i,j}$ and $m^3_{i,j}$ to $l^3_{i,j}$. Finally, for each formula F_i, we add to the graph a triangle whose vertices are labelled $o_{i,1}, o_{i,2}, F_i$ (all such triangles are disjoint). We then join vertices $o_{i,1}, o_{i,2}, F_i$ to all vertices of the form $c_{i,-}$. We will refer to the set of vertices added in this phase, relative to each formula F_i, for $1 \le i \le k-1$, as P^3_i. See Figure 6.3 for an example relative to $F_1 = (y^1_1 \vee z^1_1 \vee \neg z^2_1) \wedge (\neg z^2_1 \vee \neg z^1_1 \vee \neg y^1_1)$.

<u>Phase 4</u> We attach a copy of a K_4 to each vertex labelled F_i, where $i = 1, 2, \ldots, k-1$, by identifying F_i with one of the vertices of the clique (all

such copies are disjoint). We label the remaining 3 vertices $d_i^1, d_i^2, \overline{F_i}$. We then add to the graph (for each i) 4 vertices labelled $s_i^1, s_i^2, s_i^3, s_i^4$. We join vertices s_i^1 and s_i^2 to literal-vertex x_i and to vertex F_i. We then join s_i^3 and s_i^4 to literal-vertex $\neg x_i$ and to vertex $\overline{F_i}$. Vertex s_i^1 is also joined to s_i^2, and s_i^3 is joined to s_i^4. Finally, we join F_i and $\overline{F_i}$ to vertex f. We will refer to the set of vertices added in this phase, relative to each vertex F_i (that corresponds to formula F_i), as P_i^4. See Figure 6.4 for an example.

Phase 5 For each $r = 1, 2, \ldots, k-1$, and each variable $y \in Y_{r+1}$, there are two sets of conjunctions of literals over $Y_r \cup \{x_r\}$, called, respectively, C_y and D_y, such that for each conjunction $\alpha \in C_y$ and each conjunction $\beta \in D_y$, $(\alpha \to y)$ and $(\beta \to \neg y)$ are both clauses of F_0. The constructions relative to conjunctions of type α and of type β are very similar. For each variable y in Y_i, where $2 \leq i \leq k$, we proceed as follows. Let $y = y_i^j$, say.

- For each conjunction α, relative to y_i^j, we add to the graph the following vertices. Let $\alpha = (g_1 \wedge g_2 \wedge \ldots \wedge g_b)$, where each g_- is a literal. For each $e = 1, 2, \ldots, b$, we add to the graph a triangle (all such copies are disjoint) whose vertices are labelled $p1_i^j, p2_i^j, p3_i^j$. Note that e indicates the position of literal g_e in α. We subsequently join $p1_i^j$ and $p2_i^j$ to literal-vertex $\neg g_e$, that is, to the vertex corresponding to the negation of literal g_e in α, and join $p3_i^j$ to vertex f. For each α, relative to variable y_i^j, we then add to the graph a triangle whose vertices are labelled $p4_i^j, p5_i^j, a_i^j$ (all such copies are disjoint). We then join $p4_i^j, p5_i^j$ and a_i^j to all vertices of the form $p3_i^j$ (relative to conjunction α), and we join a_i^j to vertex f.

- For every conjunction β, relative to y_i^j, we proceed as follows. Let $\beta = (g_1 \wedge g_2 \wedge \ldots \wedge g_b)$. For each $e = 1, 2, \ldots, b$, we add to the graph a triangle (all such copies are disjoint) whose vertices are labelled $p1_i^j, p2_i^j, p3_i^j$, and join $p1_i^j$ and $p2_i^j$ to literal-vertex $\neg g_e$, that is, to the vertex corresponding to the negation of literal g_e. We then join $p3_i^j$ to vertex f. For each β, we then add to the graph a triangle whose vertices are labelled $p4_i^j, p5_i^j, b_i^j$ (all such copies are disjoint). Vertices $p4_i^j, p5_i^j$ and b_i^j are then joined to all corresponding vertices of the form $p3_i^j$ (relative to conjunction β), and vertex b_i^j is joined to f.

For $q = 2, 3, \ldots, k$, we refer to the set of vertices added in this phase relative to variables in Y_q as P_q^5. In Figure 6.4 we show an example relative to a conjunction $\alpha = (\neg y_1^1 \wedge y_1^2 \wedge x_1)$ and a clause, $((\neg y_1^1 \wedge y_1^2 \wedge x_1) \rightarrow y_2^1)$, in F_0.

Phase 6 For each vertex of the form a_-^-, a_i^j, say, corresponding to a clause $(\alpha \rightarrow y_i^j)$ in F_0, we add to the graph 2 adjacent vertices labelled, respectively, $r1_i^j$ and $r2_i^j$, and join them to a_i^j. We then join every vertex of the form $r1_i^j$ and $r2_i^j$ to literal-vertex y_i^j. For each vertex of the form b_-^-, b_i^j, say, corresponding to a clause $(\beta \rightarrow \neg y_i^j)$ in F_0, we add to the graph 2 adjacent vertices labelled, respectively, $w1_i^j$ and $w2_i^j$ and join them to b_i^j. We then join every vertex of the form $w1_i^j$ and $w2_i^j$ to literal-vertex $\neg y_i^j$. For $q = 2, 3, \ldots, k$, we refer to the set of vertices added in this phase relative to variables in Y_q as P_q^6. See Figure 6.5 for an illustration relative to: variable y_2^1, two clauses of the form $(\alpha \rightarrow y_2^1)$ and one clause of the form $(\beta \rightarrow \neg y_2^1)$. Note that, in the figure, each vertex labelled a_2^1 corresponds to a conjunction α.

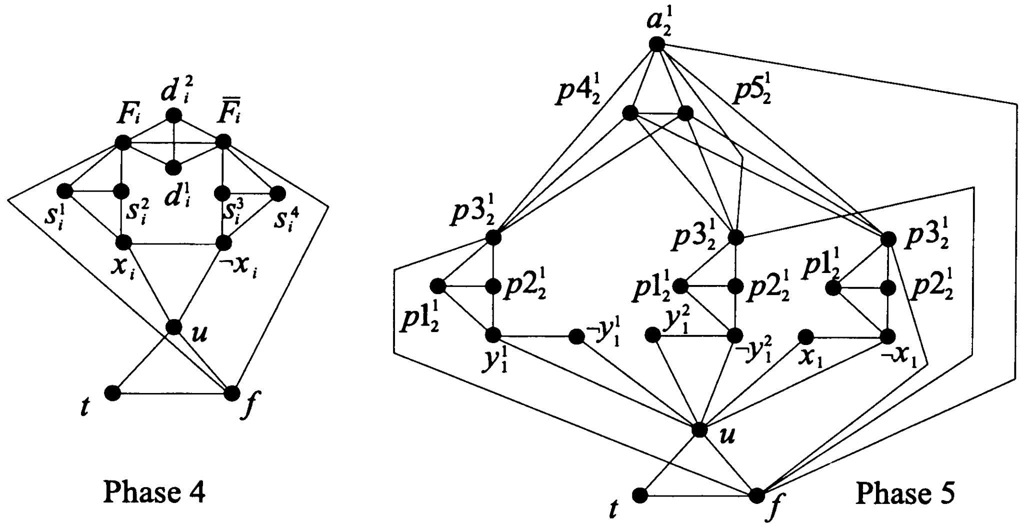
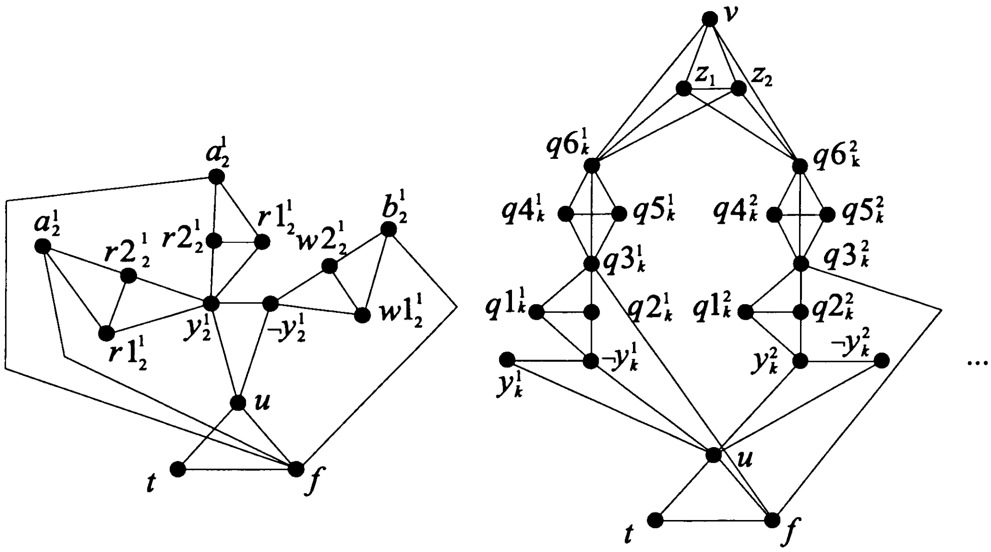


Figure 6.4: Phases 4 and 5 of the construction.

<u>Phase 7</u> For each clause in F_0 consisting of a literal from Y_k, y_k^i say, we add to the graph 6 vertices labelled, respectively, $q1_k^i, q2_k^i, q3_k^i, q4_k^i, q5_k^i$ and $q6_k^i$. We then join vertices $q1_k^i$ and $q2_k^i$ to $q3_k^i$ and y_k^i. Vertex $q3_k^i$ is then joined to vertex f, and $q1_k^i$ is joined to $q2_k^i$. Vertices $q3_k^i, q4_k^i, q5_k^i$ and $q6_k^i$ are then joined to form a K_4. Note that if the literal was of the form $\neg y_k^-$, the labelling of the added vertices would not change. We conclude the construction of graph G by adding a triangle whose vertices are labelled z_1, z_2, v. We join such vertices to all vertices of the form $q6_k^-$. We will refer to the set of vertices added in this phase as V_7. See Figure 6.5 for an illustration relative to clauses $(\neg y_k^1)$ and (y_k^2) in F_0.

Clearly the construction can be completed using logspace.



Phase 6 Phase 7

Figure 6.5: Phases 6 and 7 of the construction.

The linear ordering P is as follows:

$$u < t < f < L < V_1 < P_1^3 < P_1^4 < P_2^5 < P_2^6 < P_2^3 < P_2^4$$

$$< P_3^5 < P_3^6 < \ldots < P_{k-1}^3 < P_{k-1}^4 < P_k^5 < P_k^6 < V_7$$

We will now show that vertex v appears in the set of vertices output by the algorithm GREEDY(3-colourable) on instance (G, P, u) if, and only if, $(F_0, \ldots, F_{k-1})$ is a yes-instance of DSAT. Consider the execution of the algorithm GREEDY(3-colourable) on instance (G, P, u). The first 3 vertices in the linear ordering P are u, t and f, and it is clear that they will all be chosen by the algorithm; as they form a triangle, in any proper 3-colouring such vertices must be assigned 3 different colours. The execution continues by examining all vertices in L, that is, all literal-vertices (in any order), and they will all be chosen. As all vertices in L are adjacent to u, they must be coloured identically to either t or f.

The execution will continue by examining the vertices in V_1, that is, those vertices added to the graph during Phase 2 of the construction. First all vertices of the form $y^-_{1,1}$ and $\neg y^-_{1,1}$ will be examined (in any order) then all vertices of the form $y^-_{1,2}$ and $\neg y^-_{1,2}$ (again in any order), and finally all vertices of the form $y^-_{1,0}$ and $\neg y^-_{1,0}$ (in any order). All such vertices will be chosen, but this will have the effect that, in any proper colouring, every literal-vertex l from Y_1, such that (l) is a clause in F_0, will be coloured as t, while $\neg l$ (the literal-vertex corresponding to the negation of l) will be coloured as f. This corresponds to assigning a truth value to all variables in Y_1 that satisfies the clauses in F_0 consisting of one literal over the set of variables Y_1. The truth assignment sets the value of every variable $y \in Y_1$ to be true if the corresponding literal-vertex y is coloured with the same colour as t, and sets the value of y to be false if $\neg y$ is coloured with the same colour as t. We will denote such a truth assignment as $\tau_1(Y_1)$.

The linear ordering continues with the vertices in P_1^3, that is, the vertices added in Phase 3 relative to formula F_1. For each clause c_i in F_1 (the order in which the clauses are examined is irrelevant), the corresponding vertices in G are examined in the following order.

- First $l^1_{1,i}$, $l^2_{1,i}$ and $l^3_{1,i}$ are examined and chosen. As $l^1_{1,i}$, $l^2_{1,i}$ and $l^3_{1,i}$ are adjacent to t, they can only be coloured identically to either f or u. Note that $l^1_{1,i}$ is adjacent to the literal-vertex corresponding to the first literal in clause c_i, $l^2_{1,i}$ is adjacent to the literal-vertex corresponding to the second literal in c_i and $l^3_{1,i}$ is adjacent to the literal-vertex corresponding to the third in c_i.

- The execution continues by visiting $m^1_{1,i}$, $m^2_{1,i}$, and $m^3_{1,i}$. Vertices $m^1_{1,i}$ and $m^2_{1,i}$ are always chosen, but $m^3_{1,i}$ can be selected if, and only if, at least one of vertices $l^1_{1,i}, l^2_{1,i}, l^3_{1,i}$ (which can only be coloured as vertex u or vertex f) can be coloured as f. Note that this can only happen if at least one of the literal-vertices adjacent to $l^1_{1,i}, l^2_{1,i}$ or $l^3_{1,i}$ is coloured as t. As the vertices in V_1 have already been examined, the colouring on the literal-vertices from Y_1 is now fixed. The colouring of a chosen literal-vertex from Z_1 is still not fixed, however. This phase corresponds to checking that a satisfying truth assignment exists that satisfies clause c_i in F_1; if a colouring exists such that $m^3_{1,i}$ can be chosen then this corresponds to a satisfying truth assignment on clause c_i.

- Vertex $c_{1,i}$ is examined next, and it is chosen if, and only if, $m^3_{1,i}$ was rejected (note that this means that a corresponding satisfying truth assignment cannot exist, or vertex $m^3_{1,i}$ would have been chosen).

After all vertices corresponding to each clause in F_1, that is, all vertices of the form $l^1_{1,-}, l^2_{1,-}, l^3_{1,-}, m^1_{1,-}, m^2_{1,-}, m^3_{1,-}$ and $c_{1,-}$ have been examined, vertices $o_{1,1}$ and $o_{1,2}$ will be considered. It is clear that such vertices are always chosen. Vertex F_1 will be examined next, and it will be chosen if, and only if, none of the vertices of the form $c_{1,-}$ have previously been selected. As there is a vertex $c_{1,i}$ for each clause c_i in F_1, by assigning value true to each variable in $Y_1 \cup Z_1$ such that its corresponding literal-vertex has been coloured with the same colour as t, it would be possible to satisfy formula F_1, if vertex F_1 was chosen. If F_1 was not chosen then such a truth assignment cannot exist, or else by colouring the vertices accordingly, vertex F_1 would have been chosen. Note that, as the vertices in V_1 have previously been chosen, this phase corresponds to checking the satisfiability of $F_1(\tau_1(Y_1), Z_1)$.

The execution continues by visiting the vertices in P_1^4, that is, the vertices added in Phase 4 corresponding to formula F_1. The vertices are examined in this order: first d_1^1 then d_1^2 and finally $\overline{F_1}$. Vertex $\overline{F_1}$ can be chosen if, and only if, F_1 was rejected, because F_1, d_1^1, d_1^2 and $\overline{F_1}$ form a K_4. The execution continues by examining and choosing vertices $s_1^1, s_1^2, s_1^3, s_1^4$. This results in fixing the colouring of literal-vertices x_1 and $\neg x_1$. Vertex x_1 is coloured as t if, and only if, F_1 is chosen, while $\neg x_1$ is coloured as t if, and only if, $\overline{F_1}$ is chosen. This corresponds to deriving a truth assignment on variable x_1 according to the satisfiability of $F_1(\tau_1(Y_1), Z_1)$. Variable x_1 is set to true if literal-vertex x_1 is coloured with the same colour as t, and it is set to false if $\neg x_1$ is coloured identically to t. Note that at this stage all literal-vertices from $Y_1 \cup \{x_1\}$ have a fixed colouring. We can therefore derive a corresponding truth assignment $\tau_1(Y_1 \cup \{x_1\})$.

The execution will continue by examining the vertices in P_2^5, that is, the vertices added in Phase 5 relative to all implications $(\alpha \rightarrow y)$ and $(\beta \rightarrow \neg y)$ in F_0, for every variable $y \in Y_2$. The vertices in P_2^5 are examined in the following order. For each $j = 1, 2, \ldots, d_2$ (in any order), where d_2 is the number of variables in Y_2, first all vertices with labels of the form $p1_2^j$ are examined, then all vertices of the form $p2_2^j$ and then finally all vertices of the form $p3_2^j$ (in any order). After these vertices have been examined, all vertices of the form $p4_2^j$ and $p5_2^j$ will be examined (in any order). Finally all vertices of the form a_2^j and b_2^j will be considered (again in any order).

We will now explain how, for any variable y in Y_2, the corresponding vertices in G will be chosen or rejected. Let $y = y_2^j$, say, and consider an implication $(\alpha \to y_2^j)$. Let $\alpha = (g_1 \wedge g_2 \ldots \wedge g_l)$ (each g_- is a literal).

- For each $e = 1, 2, \ldots, l$ (in any order), the algorithm will first examine the corresponding vertices $p1_2^j$ and $p2_2^j$, and such vertices will be chosen. The corresponding vertex $p3_2^j$ is examined afterwards. Vertex $p3_2^j$ is adjacent to vertex f, and it must be coloured with the same colour as the corresponding literal-vertex $\neg g_e$ (the literal-vertex corresponding to the negation of the literal in position e in conjunction α); it follows that $p3_2^j$ can be chosen if, and only if, $\neg g_e$ (which has already been assigned a colour) has been coloured as vertex t. Note that this means that literal-vertex g_e has been coloured identically to f. If $\neg g_e$ has been coloured with the same colour as vertex f (and g_e as t) then $p3_2^j$ will be rejected.

- After all vertices of the form $p1_2^j$, $p2_2^j$ and $p3_2^j$ have been examined, the algorithm continues the execution by visiting $p4_2^j$ and $p5_2^j$, and such vertices are clearly always chosen. When vertex a_2^j is examined, it will be chosen if, and only if, none of the corresponding $p3_2^j$ have previously been chosen. Note that this means that all literal-vertices $g_1, g_2, \ldots, g_l$ have been coloured identically to vertex t.

The same holds for every implication of the form $(\beta \to \neg y_2^j)$; when vertex b_2^j is examined, it will be chosen if, and only if, all of the corresponding $p3_2^j$

have previously been rejected, which means that every literal-vertex corresponding to a literal in β has been coloured as vertex t. It follows that the derived truth assignment $\tau_1(Y_1 \cup \{x_1\})$ will satisfy a conjunction α if, and only if, the corresponding a_2^j is chosen. The same holds for all vertices b_2^j; b_2^j is chosen if, and only if, under the current truth assignment corresponding to the colouring of the vertices, the corresponding β is satisfied.

The execution continues by visiting all vertices in P_2^6, that is, all vertices of the form $r1_2^-, r2_2^-, w1_2^-$ and $w2_2^-$ added in Phase 6 (in any order). By the definition of deterministic formula, for each set of clauses of the form $(\alpha \to y)$ and $(\beta \to \neg y)$ relative to a variable y, exactly one of the conjunctions α or β evaluates to true for any truth assignment. In the graph this is reflected by the fact that, for each pair of literal-vertices corresponding to a variable in Y_2, $\{y_2^j, \neg y_2^j\}$, say, exactly one of the vertices a_2^j (each of which corresponds to a conjunction α) or one of the vertices b_2^j (each of which corresponds to a conjunction β) is chosen. This results in forcing one of the literal-vertices $y_2^j, \neg y_2^j$ to be coloured with the same colour as t. Literal-vertex y_2^j must be coloured identically to t if any of the corresponding vertices a_2^j is chosen, while vertex $\neg y_2^j$ must be coloured with the same colour as t if any of the corresponding vertices b_2^j is chosen. This is equivalent to deterministically deriving a truth assignment on all variables in Y_2. A variable y_2^j is assigned value true if literal-vertex y_2^j is coloured with the same colour as t, and it is assigned value false if literal-vertex $\neg y_2^j$ is coloured with the same colour as t. We therefore obtain a corresponding truth assignment $\tau_1(Y_1 \cup Y_2 \cup \{x_1\})$.

The execution continues by visiting all vertices in P_2^3 then P_2^4, P_3^5 and then P_3^6, that is, all vertices added in Phases 3 and 4 relative to F_2, and all vertices added in Phases 5 and 6 relative to the clauses $(\alpha \rightarrow y)$ and $(\beta \rightarrow \neg y)$ for all variables $y \in Y_3$. Then the algorithm continues its execution by visiting all vertices in P_3^3 then P_3^4, P_4^5 and then P_4^6, and so on, until the vertices in P_k^6 have been examined, which results in fixing a colouring on all literal-vertices corresponding to the variables in Y_k. Note that the ordering on the vertices in each set of the form P_i^3, P_i^4, P_{i+1}^5 and P_{i+1}^6, for $i = 2, 3, \ldots, k-1$, is the same as the one shown on the corresponding vertices in P_1^3, P_1^4, P_2^5 and P_2^6, respectively. This corresponds to deterministically deriving a truth assignment $\tau_1(Y_1 \cup \ldots \cup Y_k \cup \{x_1, \ldots, x_{k-1}\})$.

The execution terminates by examining the vertices in V_7, that is, the vertices added in Phase 7. For each clause in F_0 (in any order) consisting of a literal from Y_k, y_k^j, say, the algorithm will first visit vertex $q1_k^j$ then $q2_k^j$ and finally $q3_k^j$. As $q3_k^j$ is adjacent to f, and it must have the same colour as the corresponding literal-vertex, it follows that $q3_k^j$ can be chosen if, and only if, y_k^j has been assigned colour t. The execution proceeds by examining, and choosing, vertices $q4_k^j$ and $q5_k^j$. Then vertex $q6_k^j$ is examined, and it can be chosen if, and only if, vertex $q3_k^j$ was previously rejected. So, if vertex $q6_k^j$ is chosen, then the truth assignment corresponding to the colouring on the chosen vertices does not satisfy the clause involving variable y_k^j and vice-versa. When all vertices of the form $q1_k^-, q2_k^-, q3_k^-, q4_k^-, q5_k^-, q6_k^-$ have been examined, vertices z_1 and z_2 are examined and always chosen. Finally vertex v is examined, and it will be chosen if, and only if, all vertices of the form $q6_k^-$ have been rejected, which means that all literal-vertices from Y_k that appear in a clause from F_0 have been assigned colour t. Vertex v, therefore, can be chosen if, and only if, the corresponding derived truth assignment $\tau_1(Y_1 \cup \ldots Y_k \cup \{x_1, \ldots, x_{k-1}\})$ satisfies F_0. What is more, for $1 \leq j \leq k-1$, $F_j(\tau_1(Y_j), Z_j)$ is satisfiable if, and only if, $\tau_1(x_j) = \text{true}$. If vertex v is chosen, then $\tau_1(x_1, \ldots, x_{k-1}, Y_1, \ldots, Y_k)$ satisfies the conditions of Definition 2 of Deterministic Satisfiability, and $(F_0, \ldots, F_{k-1})$ is therefore a yes-instance of DSAT.

Conversely, if $(F_0, \ldots, F_{k-1})$ is a yes-instance of DSAT, a truth assignment $\tau(x_1, \ldots, x_{k-1}, Y_1, \ldots, Y_k)$ with the required characteristics must exist. By similar reasoning, all vertices of the form $q6_k^-$ will be rejected, and vertex v will therefore be chosen. The result follows. □

6.4 The dichotomy result

Theorem 6.2 *The problem GREEDY(linear ordering, undirected graphs, H-colourable) is* $\mathbf{P}$*-complete, if H is bipartite, and* $\mathbf{\Delta_2^P}$*-complete, if H is non-bipartite.*

Proof In the proof we will refer to the problem GREEDY(linear ordering, undirected graphs, H-colourable) as L_H. L_H can clearly be solved in $\mathbf{\Delta_2^P}$ if H is non-bipartite and in $\mathbf{P}$ if H is bipartite (the latter because L_H is exactly the same problem as the lexicographically first maximal subgraph problem for the property bipartite, and such a problem was proved $\mathbf{P}$-complete in [29].) To prove that L_H is $\mathbf{\Delta_2^P}$-complete, we will follow the strategy used by Hell and Nešetřil in [23], and which we also used in Chapter 5.

Using the constructions detailed in the previous chapter, that is, the indicator construction, the sub-indicator construction and the edge-sub-indicator construction, we obtain the following 3 lemmas.

Lemma 6.3 *If the problem $\mathcal{L}_{H^*}$ is $\mathbf{\Delta_2^P}$-complete then so is $\mathcal{L}_H$.*

Lemma 6.4 *If the problem $\mathcal{L}_{\tilde{H}}$ is $\mathbf{\Delta_2^P}$-complete then so is $\mathcal{L}_H$.*

Lemma 6.5 *If the problem $\mathcal{L}_{\hat{H}}$ is $\mathbf{\Delta_2^P}$-complete then so is $\mathcal{L}_H$.*

The proof of the lemmas follows immediately from the proofs of the corresponding 3 lemmas in Chapter 5, by simply encoding our linear ordering as a directed graph consisting of an Hamiltonian path.

To conclude the proof we proceed exactly as we did in the previous chapter, with the only difference being that our complete problem is now GREEDY(linear ordering, undirected graphs, 3-colourable). The result follows. □

6.5 Conclusion

In this chapter we showed a dichotomy result for the problem GREEDY(linear ordering, undirected graphs, H-colourable), and this concludes our study of the complexity of the problem GREEDY(ordering, $\mathcal{C}$, π).

In the following chapter we will introduce a new general greedy algorithm, called MaxDegree(π). Like we did in the case of GREEDY(π), we will examine the complexity of the related problem MaxDegree($\mathcal{C}$, π) and we will obtain new complete problems for the classes **NP** and $\mathbf{\Sigma_2^P}$.

Chapter 7

MaxDegree

7.1 Introduction

In the previous chapters we discussed, for different values of the parameters: ordering, $\mathcal{C}$ and π, the complexity of the problem GREEDY(ordering, $\mathcal{C}$, π), and we have obtained new problems complete for the classes $\mathbf{P}, \mathbf{NP}$, $\mathbf{\Delta_2^p}$ and $\mathbf{\Sigma_2^p}$.

In this final chapter we will show that the techniques used to obtain these results can be applied again with only relatively simple modifications to problems relative to another greedy algorithm, MaxDegree(π).

We will begin the chapter by defining our new greedy algorithm, we will then move on to obtain a new class of **NP**-complete problems and we will finish the chapter by proving the completeness of a problem for $\mathbf{\Sigma_2^p}$.

7.2 MaxDegree(π)

Given an undirected graph $G = (V, E)$, where $|V| = n$, removing a vertex of highest degree from G, and then recursively applying the same procedure to the obtained subgraph until the empty graph is obtained, gives rise to an elimination sequence on V. We can think of any such sequence as a linear order $v_1 <_s v_2 <_s \ldots <_s v_n$ on V. Greenlaw [17] proved that, given an undirected graph $G = (V, E)$ and 2 vertices x_1, x_2 in G, it is **NP**-complete to decide whether there is an elimination sequence on V such that $x_1 <_s x_2$. He also proved that if the vertices of G are numbered $1, 2, \ldots, n$, and at any stage the removed vertex is the smallest numbered vertex of highest degree, deciding whether there exists an elimination sequence on V such that $x_1 <_s x_2$ is **P**-complete. Inspired by such results, we will consider the complexity of deciding whether, given an undirected graph G and a vertex v in G, v appears in any of the lexicographically first maximal subgraphs of G, satisfying a certain property π, when the linear order on the vertices of G is given by any of the elimination sequences on V.

Let π be a property on graphs, and let G be an undirected graph. The algorithm MaxDegree(π) is as follows:

```
input(G)
  S := ∅
  W := G
  while W ≠ ∅ do
    current-vertex := a vertex of highest degree in W
    if π(S∪{current-vertex},G) then
```

```
        S := S∪{current-vertex}
        fi
      remove current-vertex and its incident edges from W
      od
  output(S)
```

The execution of the algorithm begins by creating a copy, called W, of the input graph G. At every execution of the while loop, a vertex of highest degree in W is chosen, and the subgraph of G induced by the set of vertices $S \cup \{current-vertex\}$ is tested for property π. If the subgraph satisfies π then *current-vertex* is added to the set S, else it is rejected. Regardless of whether *current-vertex* is selected or not, it is subsequently removed from graph W, and the execution continues with another repetition of the while loop. The program terminates when the graph W does not contain any more vertices. The algorithm MaxDegree(π) is nondeterministic and it outputs sets of vertices that induce, if the property π is hereditary, maximal subgraphs of the input graph G that satisfy property π.

Let $\mathcal{C}$ be a class of graphs and let π be some property on graphs. The problem MaxDegree($\mathcal{C}, \pi$) has as its instances tuples (G, v), where G is a graph from the class $\mathcal{C}$ and v is a vertex in G. A yes-instance of the problem is an instance for which there exists an execution of the algorithm MaxDegree(π) on input G that results in vertex v being output.

We stress here that MaxDegree(π) is a nondeterministic algorithm because there is no rule on how the algorithm should choose the next vertex if there is more than one vertex of highest degree in the graph W. If we give a heuristic that always makes such a decision, the algorithm becomes deterministic.

One possible such heuristic is the following. Let us assume that the vertices of our instance graph are labelled each with a different natural number between 1 and n, where n is the number of vertices. We can therefore obtain the following algorithm, which we will call DetMaxDegree(π).

```
input(G)
  S := ∅
  W := G
  while W ≠ ∅ do
    current-vertex := the smallest numbered vertex
    of highest degree in W
    if π(S∪{current-vertex},G) then
      S := S∪{current-vertex}
      fi
    remove current-vertex and its incident edges from W
    od
output(S)
```

The algorithm always chooses the vertex with highest degree and smallest numbered label and, if the property π is hereditary, it outputs a set of vertices that induce a maximal subgraph of the input graph G satisfying the property π. We can define the problem DetMaxDegree($\mathcal{C}$, π) as we did in the case of the algorithm MaxDegree(π). We will now show that the problem DetMaxDegree(undirected graphs, π) is **P**-complete for any property π which is hereditary, non-trivial on undirected graphs and testable in polynomial time (via a logspace reduction).

Theorem 7.1 *Let π be a polynomial time testable, hereditary graph property non-trivial on undirected graphs. The problem DetMaxDegree(undirected graphs, π) is complete for* $\mathbf{P}$.

Proof For any property π testable in polynomial time, hereditary and non-trivial on undirected graphs, the problem DetMaxDegree(undirected graphs, π) is clearly in $\mathbf{P}$. By [29], we know that the problem (from now on called LFMSP(π)) of deciding whether a given vertex of a given undirected graph G, whose vertices are linearly ordered, lies in the lexicographically first maximal subgraph of G satisfying π is $\mathbf{P}$-complete. For any property π satisfying the aforementioned conditions, we reduce an instance (G, v) of the problem LFMSP(π), where: $G = (V, E)$ is an undirected graph, $V = \{1, 2, \ldots, n\}$ and $v \in V$, to an instance (G', v') of the problem DetMaxDegree(undirected graphs, π), such that (G, v) is a yes-instance of LFMSP(π) if, and only if, (G', v') is a yes-instance of DetMaxDegree(undirected graphs, π).

We begin the construction of G' with a copy of G and we take v' to be the vertex previously known as v: we refer to the vertices of G' that belong to the copy of G as G-vertices. In order to force the algorithm DetMaxDegree(π) to examine the vertices of G' following the linear ordering on G, we identify each vertex in G' with the centre of a star. The size of each star depends on the position of the vertex in the linear ordering. Let $p(x)$ denote the position of a vertex x in the linear ordering: the size of the star attached to the G-vertex x in G' will be $n(n-p(x))$. So, for example, if x is the first vertex in the linear ordering then $p(x) = 1$, and the size of the added star is $n(n-1)$. It is clear that all the G-vertices will be examined before any other vertex in G', because they are at any stage the vertices of highest degree in W (the copy of G' created by the algorithm). The order in which the G-vertices will be visited by DetMaxDegree(π) is given by the linear ordering in G, and therefore if a vertex in G appears in the lexicographically first maximal subgraph of G, then the corresponding G-vertex in G' will be output by the execution of DetMaxDegree(π) on instance G', because the subgraph of G' induced by the G-vertices is exactly graph G. As vertex v' is the G-vertex corresponding to vertex v in G, it will be chosen if v appears in the lexicographically first maximal subgraph of G satisfying property π. Note that at any stage in the execution of the algorithm the vertices of G' that are not G-vertices have degree at most one, and will therefore be examined after every G-vertex has been examined. It follows that they will not affect the choice or rejection of vertex v'. By similar reasoning it is not difficult to see that if vertex v' is output by DetMaxDegree(π) then vertex v will also appear in the lexicographically first maximal subgraph of G satisfying π. As the reduction can clearly be performed using logspace, the result follows. □

In the next section we will examine the complexity of the problem MaxDegree(undirected graphs, π) for properties π which are testable in polynomial time, hereditary, non-trivial on the class of undirected graphs and satisfied by all independent sets of vertices. Paralleling the results shown in Chapter 3, we will obtain a class of **NP**-complete problems. To obtain this result, we will begin by proving a problem complete for **NP**, and then use it as a base of a reduction to obtain our main result.

7.3 Another class of NP-complete problems

We will begin the section by proving our base case problem, that is, we will prove the completeness for **NP** of the problem MaxDegree(undirected graphs, independent set).

Theorem 7.2 *The problem MaxDegree(undirected graphs, independent set) is* **NP**-*complete.*

Proof As the property "independent set" is testable in polynomial time, the problem MaxDegree(undirected graphs, independent set) is clearly solvable in **NP**. To show completeness we will reduce from the **NP**-complete problem 3-SAT, defined in Section 4.2.

From an instance (C, X) of 3-SAT with m clauses and n variables, we construct an instance (G, v) of MaxDegree(undirected graphs, independent set) as follows. For each literal $l_{i,j}$ appearing in clause c_i, where $j \in \{1, 2, 3\}$, there are 2 corresponding vertices in G, labelled $l_{i,j}$ and $\bar{l}_{i,j}$, respectively, corresponding to the literal and to its negation. We will refer to such vertices as *literal-vertices* and *negation-vertices* respectively. We join any two vertices that correspond to literals which are the negations of each other; so for example, all vertices labelled x_1 will be joined to all vertices labelled $\neg x_1$. For each clause i, where $1 \leq i \leq m$, we add to the graph a corresponding vertex labelled o_i adjacent to $l_{i,1}$, $l_{i,2}$, and $l_{i,3}$. We will refer to such vertices as *o-vertices*. We join all o-vertices to a newly added vertex, which we choose to be v. The construction is concluded by adding to the graph a series of stars, as follows. To each literal-vertex $l_{i,j}$, corresponding to a literal in a clause, we attach a $(m+5)$-star by identifying $l_{i,j}$ with the centre of the star (m is the number of clauses in C). To each negation-vertex $\bar{l}_{i,j}$ corresponding to the negation of a literal in one of the clauses, we attach a $(m+6)$-star by identifying $\bar{l}_{i,j}$ with the centre of the star. Finally we attach a copy of a m-star to each o-vertex. See Figure 7.1 for an example relative to instance $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_5 \vee x_4 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_4)$ of 3-SAT.

We will now show that (C, X) is satisfiable if, and only if, vertex v appears in one of the sets of nodes output by the algorithm MaxDegree(independent set) on instance G.

Suppose that (G, v) is a yes-instance of MaxDegree(undirected graphs, independent set), that is, vertex v appears in at least one of the sets of vertices output by an execution of MaxDegree(independent set) on instance G. We will show that we can derive a truth assignment that satisfies all the clauses of (C, X) from the set of vertices output by the algorithm.

The algorithm always examines a vertex of highest degree in W (the graph obtained from a copy of G). In W every literal-vertex and every negation-vertex has degree at least $m+6$, while every other vertex has lower degree. It follows that every execution will start by choosing a literal- or a negation-vertex.

By construction every literal- and every negation-vertex is adjacent to all vertices that correspond to its negation; therefore if a vertex labelled x_i, say, is examined and chosen then all literal-vertices and negation-vertices corresponding to its negation, that is, all vertices labelled $\neg x_i$, will subsequently be rejected, while every vertex labelled x_i will be chosen. At the beginning of

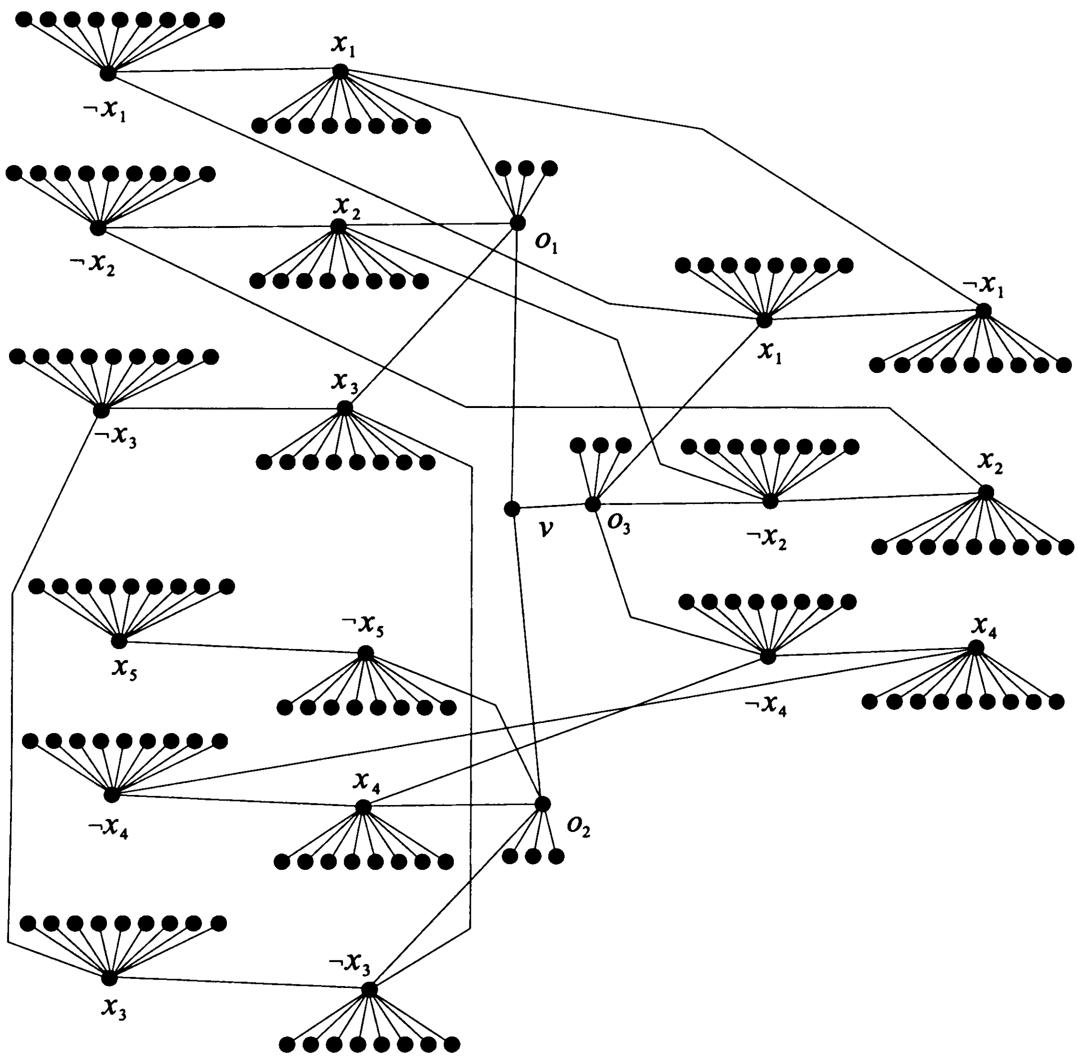


Figure 7.1: The graph G corresponding to $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_5 \vee x_4 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_4)$.

the execution, every two vertices labelled x_i and $\neg x_i$ have the same degree, and the algorithm will therefore nondeterministically choose one of them. This corresponds to assigning a truth value to variable $x_i \in X$: true if x_i is selected, and false if $\neg x_i$ is selected. The execution will proceed by examining all literal- and negation-vertices, as they are, at any stage, the vertices of highest degree in W. The examination of all literal- and negation-vertices therefore corresponds to guessing a truth assignment on all the variables in X. The algorithm will then examine all o-vertices, as they now have degree $m + 1$ in W, while every other vertex has smaller degree. Every o-vertex will be examined and it will be chosen if, and only if, none of its adjacent literal-vertices have previously been selected. After all o-vertices have been examined, graph W consists of an independent set, and all the vertices that constitute it will be examined one after the other. All vertices that were leaves of a star will be rejected if the centre of the corresponding star was previously chosen, and they will be selected otherwise. Vertex v will be selected if, and only if, all o-vertices were rejected. We assumed that v was selected, which means that none of the o-vertices were chosen; which also means that for every o-vertex, at least one literal-vertex adjacent to it has been selected. The truth assignment that corresponds to the chosen literal-vertices appearing in G, satisfies each clause in C.

Conversely, suppose that (C, X) is satisfiable by some truth assignment t. An execution of the algorithm will select literal-vertices and negation-vertices that correspond to t and, as such an assignment satisfies (C, X), this will result in every o-vertex being rejected. When vertex v is examined, it follows that it will be chosen by the algorithm, because none of its neighbours appears in S. The construction can clearly be completed in logspace, and the result therefore follows. $\square$

We can now prove our main result of the chapter by reducing from the problem MaxDegree(undirected graphs, independent set).

Theorem 7.3 *Let π be a graph property that is polynomial time testable, hereditary, non-trivial on undirected graphs and satisfied by all sets of independent vertices. The problem MaxDegree(undirected graphs, π) is complete for* **NP**.

Proof To prove the theorem we will use a technique similar to the one used in Theorem 3.7. If π is a polynomial time testable property then the problem MaxDegree(undirected graphs, π) is clearly solvable in **NP**. For the definition of α- and β-sequences see Section 3.4. By assumption property π is non-trivial on graphs, therefore there must be (at least) one graph with smallest β-sequence amongst all graphs that violate π. We will refer to such a graph as J.

$$\beta_J = \min\{\beta_G : G \text{ is a graph violating } \pi\}.$$

Let $J_1, J_2, \ldots, J_k$ be the connected components of J ordered according to $\alpha_{J_1} \geq_L \alpha_{J_2} \geq_L \ldots \geq_L \alpha_{J_k}$. It follows that J has β-sequence $\beta_J = (\alpha_{J_1}, \alpha_{J_2}, \ldots, \alpha_{J_k})$. Let $c = c_{J_1}$ (the cut point relative to α_{J_1}) and let the connected components of J_1 relative to c be $I_0 \cup \{c\}, I_1 \cup \{c\}, \ldots, I_m \cup \{c\}$, where $|I_0| \geq |I_1| \geq \ldots \geq |I_m|$. Denote by I_* the subgraph of J_1 induced by the vertices of $I_1 \cup \ldots \cup I_m$.

Property π is, by definition, satisfied by all independent sets of vertices; we therefore obtain without loss of generality that $\langle I_0 \cup \{c\}\rangle_J$ must contain at least one edge (or J would be an independent set, and it would not violate π).

To prove the **NP**-completeness of the problem MaxDegree(undirected graphs, π), we reduce from the problem MaxDegree(undirected graphs, independent set). From an instance (G, v) of MaxDegree(undirected graphs, independent set), we derive an instance (G', v') of MaxDegree(undirected graphs, π) with the appropriate properties.

Choose d to be any vertex of I_0 adjacent to c in J; let $s = \max\{\deg_J(c), \deg_J(d)\}$, let $f = s\Delta(G) + \deg_J(c) + 1$ and let $e = \max\{f, \Delta(J)\}$ (these numbers are used later in the construction).

We will refer as I_{*-e} to the graph obtained by adding one copy of an e-star to each vertex in I_* by identifying such a vertex with the centre of the star (all such copies are disjoint). To define J_{i-e}, where $i = 2, 3, \ldots, k$, we will use the same strategy with graphs $J_2, J_3, \ldots, J_k$, that is, we will add to each vertex p in J_i one copy of an e-star by identifying p with the centre of the star to obtain J_{i-e}. We obtain I_{0-e} from I_0 by adding to each vertex p in $I_0 \setminus \{d\}$ a copy of an e-star as previously explained. Notice that I_{0-e} contains vertex d.

We will divide the construction of G' from G in several steps.

<u>Phase 1</u> For each vertex u of G, we attach a copy of $\langle I_* \cup \{c\}\rangle_J$ by identifying u with c (all such copies are disjoint). Call the resulting graph $\tilde{G}$. Note that the vertex set of $\tilde{G}$ consists of the vertices of G, which we call the *G-vertices*,

together with disjoint copies of the vertices of I_*.

<u>Phase 2</u> We replace each edge (u,v) of $\tilde{G}$, where u and v are G-vertices, by a copy of $\langle I_0 \cup \{c\}\rangle_J$ by identifying u with c and v with d (all such copies are disjoint). Notice that, as vertices c and d are adjacent in $\langle I_0 \cup \{c\}\rangle_J$, vertices u and v are also adjacent in G'.

<u>Phase 3</u> We replace each copy of I_* added in Phase 1 with a copy of I_{*-e}, and replace each copy of I_0 added in Phase 2 with a copy of I_{0-e}. This means that each vertex in a copy of I_* and each vertex in a copy of $\langle I_0 \setminus \{d\}\rangle_J$ is now the centre of an e-star.

<u>Phase 4</u> We add disjoint copies of $J_{2-*}, J_{3-*}, \ldots, J_{k-*}$ to obtain G', and we choose v' to be the G-vertex which was previously known as v in G.

See Figure 7.2 for an example.

By construction, the degree of every vertex which is the centre of an e-star is, at any stage of the execution of the algorithm, higher than any other vertex in W' (the copy of graph G' generated by MaxDegree(π)). We will refer to the set of vertices which are the centre of an e-star as S_0. As the algorithm always chooses a vertex of highest degree, it follows that all the vertices in S_0 will be examined first. The subgraph of G' induced by S_0 has the form of graph K of Lemma 3.8, and all such vertices will therefore be chosen by every execution of MaxDegree(π) on instance G'.

Graph W', after the removal of all vertices in S_0, consists of a copy of graph G plus an independent set of vertices (the leaves of the added e-stars). Therefore if for some execution of MaxDegree(independent set) on

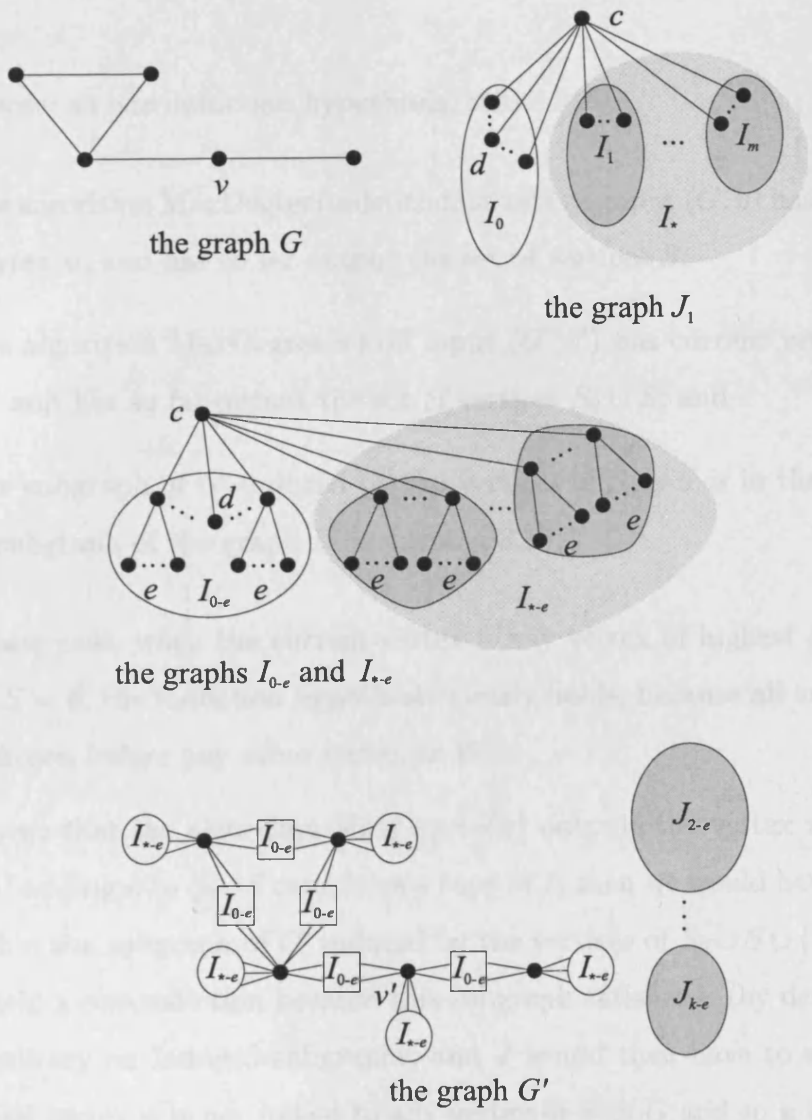


Figure 7.2: The construction of G' from G.

instance (G, v) vertex g_1 is examined before g_2 then for some execution of MaxDegree(π) on instance (G', v'), G-vertex g_1 will be examined before G-vertex g_2.

Suppose, as our induction hypothesis, that:

- the algorithm MaxDegree(independent set) on input (G, v) has current-vertex u, and has so far output the set of vertices S;
- the algorithm MaxDegree(π) on input (G', v') has current vertex u in G' and has so far output the set of vertices $S_0 \cup S$; and
- the subgraph of G' induced by the vertices of $S_0 \cup S$ is in the form of a subgraph of the graph N in Lemma 3.10.

In the base case, when the current-vertex is any vertex of highest degree in W, and $S = \emptyset$, the induction hypothesis clearly holds, because all vertices in S_0 are chosen before any other vertex in W'.

Suppose that the algorithm MaxDegree(π) outputs the vertex u. If u is such that adding u to $S_0 \cup S$ completes a copy of I_0 then we would have a copy of J within the subgraph of G' induced by the vertices of $S_0 \cup S \cup \{u\}$. This would yield a contradiction because this subgraph satisfies π (by definition), π is hereditary on induced subgraphs, and J would then have to satisfy π. Hence, the vertex u is not joined to any vertex of S in G and so u is output by the algorithm MaxDegree(independent set).

Conversely, if the algorithm MaxDegree(independent set) outputs u then this is because $S \cup \{u\}$ is an independent set in G; and consequently $S_0 \cup$

$S \cup \{u\}$ induces in G' a subgraph of the form of a subgraph of the graph N in Lemma 3.10. Hence, by Lemma 3.10, u is output by the algorithm MaxDegree(π).

Every execution of MaxDegree(π) will terminate by examining the vertices which were previously leaves of some e-star (and that now have degree zero in W' because the centres of the respective stars have already been examined). Such vertices might or might not be chosen, but this will not affect the outcome of the execution, as vertex v' will have already been examined. We refer to the set of chosen vertices from the leaves of the stars as S_1: note that S_1 might be empty.

By induction, we obtain that if S is a set of vertices output by the algorithm MaxDegree(independent set) on input (G, v) then $S_0 \cup S_1 \cup S$ is output by the algorithm GREEDY(π) on input (G', v'), and conversely. As the construction can clearly be carried out in logspace, the result follows. $\square$

In the next section we will abandon the requirement that our property π is testable in deterministic polynomial time, and examine the complexity of the problem MaxDegree(undirected graphs, π) for a graph theoretical property π testable in **NP**.

7.4 Another Σ_2^p-complete problem

In this section we will show that the problem MaxDegree(undirected graphs, π) considered in the setting of **NP** testable properties π, is not solvable in **NP** any more but is instead solvable in $\mathbf{\Sigma_2^p}$. The techniques used in Chapters

5 and 6 to prove the complexity of the problem when we take our property π to be H-colourable do not appear to work in this setting, and therefore we did not manage to prove a general result as in the case of polynomial time testable properties. Nevertheless we showed the completeness of a specific problem for the complexity class $\mathbf{\Sigma_2^P}$.

Theorem 7.4 *The problem MaxDegree(undirected graphs,* 3*-colourable) is* $\mathbf{\Sigma_2^P}$*-complete.*

Proof In this proof we will refer to the problem MaxDegree(undirected graphs, 3-colourable) as $\mathcal{D}$. It is clear that $\mathcal{D}$ can be solved in $\mathbf{\Sigma_2^P}$; to prove the completeness of $\mathcal{D}$ for $\mathbf{\Sigma_2^P}$ we will reduce from the problem NOT CERTAIN 3-COLOURING OF BOOLEAN EDGE-LABELLED GRAPHS, which will be abbreviated as $\mathcal{N}$. Problem $\mathcal{N}$ was defined in the proof of Theorem 5.1.

Given an instance Z of $\mathcal{N}$, we shall construct an instance (G, v) of $\mathcal{D}$, where G is an undirected graph and v is a distinguished vertex of G. Moreover, Z will be a yes-instance of $\mathcal{N}$ if, and only if, (G, v) is a yes-instance of $\mathcal{D}$; and the construction will be such that it can be completed using logspace. The reduction follows very closely the schema seen in Theorem 5.1, with the main difference being that to force vertices to be examined in a certain order we will increase their degree by identifying them with the centres of stars of different sizes.

Let $Z = (U, F)$ and suppose that $U = \{1, 2, \ldots, n\}$. We build the undirected graph G from Z as follows.

(a) For each vertex $i \in U$, 'attach' a copy of K_4 by identifying vertex i

with one of the vertices of the clique. Denote the other three vertices by a_i, b_i^1 and b_i^2. We refer to the original vertices of U as *Z-vertices*, the vertices of $\{a_i : i = 1, 2, \dots, n\}$ as *a-vertices* and the vertices of $\{b_i^1, b_i^2 : i = 1, 2, \dots, n\}$ as *b-vertices.*

(b) Retain any unlabelled edge (i, j) of F (between Z-vertices i and j).

(c) For any labelled edge (i, j) of F (between Z-vertices i and j), where $i < j$ and where the label is $L_{i,j}^1 \vee L_{i,j}^2$ ($L_{i,j}^1$ refers to the first literal labelling edge (i, j) and $L_{i,j}^2$ to the second), replace the edge with a copy of the graph G_1 shown in Figure 7.3. The vertices of $\{\bar{L}_{i,j}^1, \bar{L}_{i,j}^2 : (i, j) \in F,$ where $i < j\}$ are called *L-vertices.* Every L-vertex of any G_1 has an associated literal, *e.g.*, if the literal $L_{4,6}^1 = \neg X_{3,2}$ then the associated literal of vertex $\bar{L}_{4,6}^1$ is $X_{3,2}$, that is, the negation. So vertices $\bar{L}_{i,j}^1$ and $\bar{L}_{i,j}^2$ in G_1 correspond, respectively, to the negation of $L_{i,j}^1$ and $L_{i,j}^2$. Notice that an L-vertex of a copy of G_1 might have the same associated literal as an L-vertex of another copy of G_1. The vertices of $\{c_{i,j} : i, j = 1, 2, \dots, n\}$ are called *c-vertices*, the vertices of $\{d_{i,j} : i, j = 1, 2, \dots, n\}$ are called *d-vertices* and the vertices of $\{e_{i,j}^1, e_{i,j}^2 : i, j = 1, 2, \dots, n\}$ are called *e-vertices.*

(d) Include a disjoint copy of K_4, whose vertices are $\{y, z, w, v\}$ and join vertices y, z and w to every a-vertex.

(e) For every variable $X_{i,j}$ such that at least one occurrence of $X_{i,j}$ or $\neg X_{i,j}$ appears as a label of an edge in Z, construct the graph (from now on called the *variable-graph*) shown in Figure 7.3. Note that in every such graph the number of pairs of vertices labelled $X_{i,j}^1$ and $\neg X_{i,j}^1$

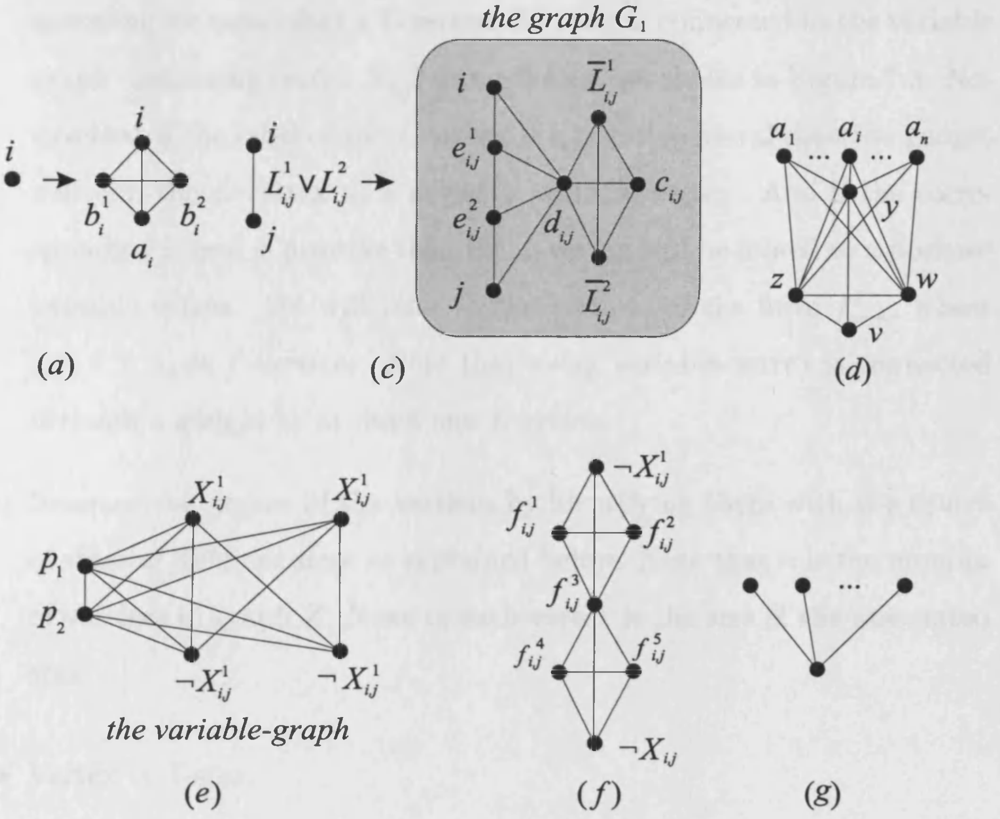


Figure 7.3: The phases of the construction of G from Z.

is determined by the number of occurrences of literals labelled $X_{i,j}$ or $\neg X_{i,j}$ in Z: there is a pair for each occurrence. We will refer to the vertices labelled p_1 and p_2 as *p-vertices*, and to the vertices of the form $X^1_{\_,\_}$ and $\neg X^1_{\_,\_}$ as *variable-vertices*. Variable-vertices will be considered positive if they are of the form $X^1_{\_,\_}$, and negative if of the form $\neg X^1_{\_,\_}$.

(*f*) Connect every L-vertex to the corresponding variable-graph (by corresponding we mean that a L-vertex, $X_{i,j}$ say, is connected to the variable graph containing vertex $X^1_{i,j}$) using the gadget shown in Figure 7.3. Notice that if the label of the L-vertex is a negative literal then the gadget will join the L-vertex to a negative variable-vertex. And if the corresponding literal is positive then the L-vertex will be joined to a positive variable-vertex. We will refer to the vertices of the form $f^i_{-,-}$, where $1 \leq i \leq 5$, as *f-vertices*. Note that every variable-vertex is connected through a gadget to at most one L-vertex.

(g) Increase the degree of the vertices by identifying them with the centre of stars of different sizes as explained below. Note that n is the number of vertices in graph Z. Next to each vertex is the size of the associated star.

- Vertex v: 1-star.
- Vertices $\{z, y, w\}$: 2-star.
- a-vertices: $(n + 5)$-star.
- b-vertices: $(n + 11)$-star.
- Z-vertices: $(n + 15)$-star.
- e-vertices: $(3n + 16)$-star.
- d-vertices: $(3n + 17)$-star.
- c-vertices: $(3n + 24)$-star.
- L-vertices: $(3n + 27)$-star.

- Vertices $f^4_{-,-}$ and $f^5_{-,-}$: $(3n+33)$-star.
- Vertices $f^3_{-,-}$: $(3n+36)$-star.
- Vertices $f^1_{-,-}$ and $f^2_{-,-}$: $(3n+42)$-star.
- Variable-vertices: $(3n+45)$-star if the vertex is adjacent to some f-vertices: $(3n+48)$-star otherwise.
- p-vertices: $(n^2+3n+50)$-star.

We give an example of the construction of (G, v) from Z in Figure 7.4 (note that to avoid cluttering the figure we did not label all the vertices nor add the stars as detailed in phase g). Suppose that Z is a yes-instance of problem $\mathcal{N}$. Hence there exists a truth assignment t such that $t(Z)$ is not 3-colourable. By construction of the graph G, at the beginning of the execution of the algorithm the vertices of highest degree in W (the copy of G) are the p-vertices, and they will be examined (and chosen) before any other vertex in the graph. The execution of the algorithm will then continue by examining all the variable-vertices, as after the removal of the p-vertices from W, they are now the vertices of highest degree. Every positive variable-vertex forms a copy of a K_4 with the vertices p_1, p_2 and with each negative variable-vertex in the corresponding variable-graph: it is therefore clear that either the positive or the negative variable-vertices can be chosen, but not both. For any variable-graph, for any execution of the algorithm exactly one of the groups of positive or negative variable-vertices will be chosen, and such a choice will correspond to a truth assignment for the variable. Notice that, in any variable-graph, each variable-vertex has the

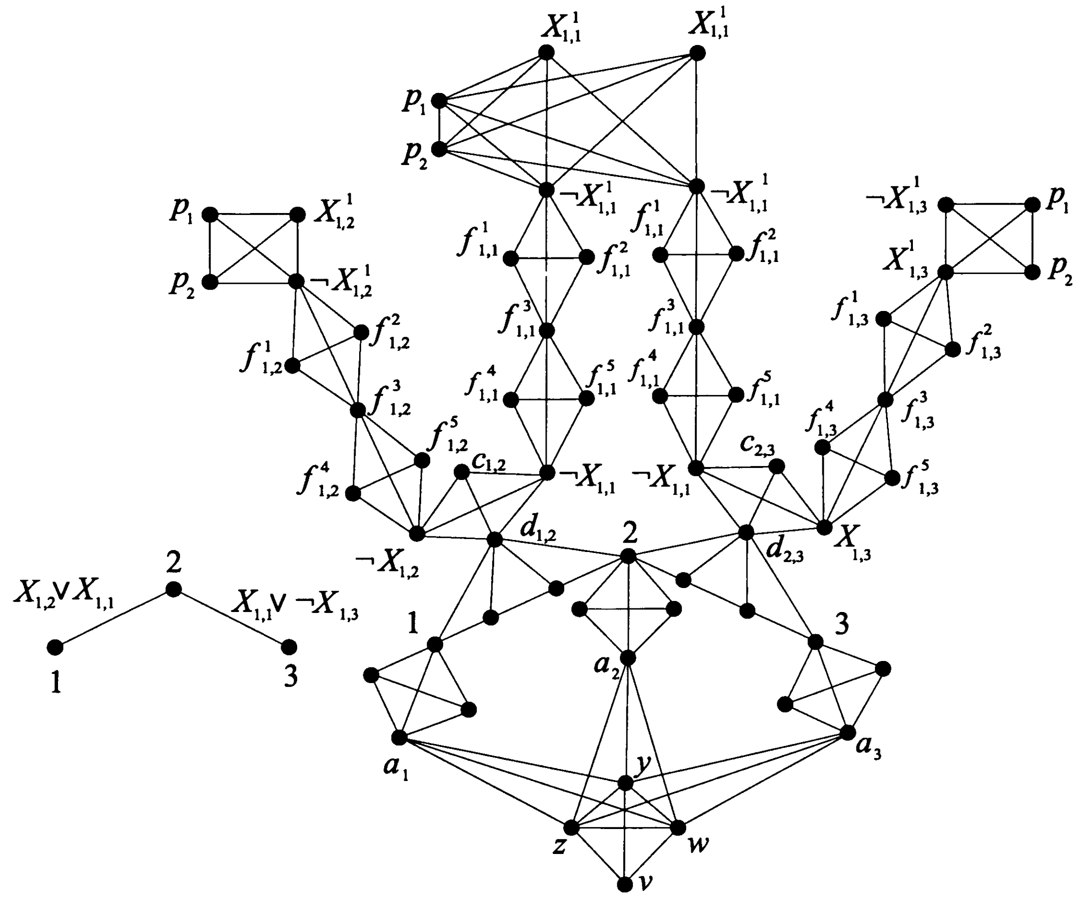


Figure 7.4: The construction of G from Z.

same degree, so they could potentially all be the first vertex chosen by an execution of the algorithm. We can therefore consider the execution of the algorithm MaxDegree(3-colourable) where the set of chosen variable-vertices corresponds to the truth assignment t.

After all p- and variable-vertices have been considered, and removed from W, the vertices of highest degree in W are the f-vertices, and they will all be examined before any other remaining vertex. First all vertices of the form $f^1_{-,-}$ and $f^2_{-,-}$ are examined, and always chosen. Then all vertices labelled $f^3_{-,-}$ are chosen if, and only if, the corresponding variable-vertices, that is the ones that form a K_4 with them, have been rejected. Finally all vertices of the form $f^4_{-,-}$ and $f^5_{-,-}$ are examined (because of their degree in W), and always selected by every execution of the algorithm.

The algorithm will then continue the execution by visiting the L-vertices, as they are now the vertices of highest degree in W. It is straightforward to notice that any such vertex can be chosen if, and only if, the corresponding variable-vertex was previously chosen. At this point the algorithm will examine all c-vertices, and they will clearly be chosen in every execution of the algorithm.

The vertices of highest degree in W are now the d-vertices. Let us freeze the execution at this point. Note that if the truth assignment t makes the label of some edge (i, j) of F true then, at our freeze-point, the vertex $d_{i,j}$ is adjacent to at most 2 vertices of S (the set of vertices chosen so far), and so this vertex $d_{i,j}$ is subsequently output by MaxDegree(3-colourable).

Conversely, if the truth assignment t makes the label of some edge (i, j) of F false then, at our freeze-point, the vertex $d_{i,j}$ is adjacent to 3 mutually adjacent vertices of S and so this vertex $d_{i,j}$ is not subsequently output by MaxDegree(3-colourable). After all d-vertices have been examined, the vertices of higher degree are the e-vertices, and they will be chosen by every execution of the algorithm. The vertices of higher degree are now the Z-vertices, and they will be examined by the algorithm next. Let (i, j) be some edge of Z which is either unlabelled or whose label has been made true by t. It may or may not be the case that the vertices i and j are output; but if they are both output then at the point after the second of these vertices is output, the subgraph induced by the vertices of S can be 3-coloured but not so that i and j have the same colour. This is so because each of the vertices $d_{i,j}$, $e^1_{i,j}$ and $e^2_{i,j}$ is in S. Hence, as we know that $t(Z)$ cannot be 3-coloured, there must be some Z-vertex that is not output. The algorithm will then examine all the b-vertices and, subsequently, all the a-vertices because they are now the vertices of highest degree. Clearly all the b-vertices will be chosen. Every a-vertex can only be chosen if the corresponding Z-vertex is rejected, therefore it follows that there is at least one a-vertex output. The vertices of highest degree in W are now y, z and w and they will therefore be examined next (they all have the same degree, but the order in which they are examined is not important). The first two vertices to be examined will be chosen, while the third will be rejected, as at least one a-vertex has been chosen. At this point the vertex of highest degree in W is vertex v and, as only two of y, z, w have been chosen, vertex v will be selected as well. Hence, (G, v) is a yes-instance of problem $\mathcal{G}$. The algorithm will terminate its execution by examining the leaves of the added stars: their choice or rejection will not affect the outcome of the execution.

Conversely, suppose that (G, v) is a yes-instance of problem $\mathcal{G}$. Fix an accepting execution of the algorithm MaxDegree(3-colourable) on input (G, v) and denote the truth assignment given by the chosen variable-vertices by τ. This execution gives rise to a truth assignment t on the literals labelling the edges of the graph Z: if τ is such that a positive variable-vertex, with label $X_{i,j}$, say, is chosen then set $t(X_{i,j})$ to be true; and if τ is such that a negative variable-vertex, with label $\neg X_{i,j}$, say, is chosen then set $t(X_{i,j})$ to be false (note that this truth assignment is well-defined). By arguing as we did earlier, for any $i, j \in \{1, 2, \ldots, n\}$ with $i < j$ and where (i, j) is a labelled edge of Z, the truth assignment t makes $L^1_{i,j} \vee L^2_{i,j}$ true if, and only if, the vertices $d_{i,j}$, $e^1_{i,j}$ and $e^2_{i,j}$ are output.

At various points in the execution of MaxDegree(3-colourable), a check is made to see whether the vertices of S induce a 3-colourable graph. Consider such a check and suppose that the vertices of $\{d_{i,j}, e^1_{i,j}, e^2_{i,j}\}$ have been placed in S. Consider the subgraph K of G induced by those vertices that are both in S and in the copy of G_1 pertaining to the labelled edge (i, j) of Z. In particular, consider the role of K when it comes to attempting to colour the subgraph of G induced by the vertices of S. A simple combinatorial verification yields that the role of the vertices of K is to allow i and j to be coloured with any pair of distinct colours but not with identical colours. Hence, any check to see whether the subgraph of G induced by the vertices of S can be 3-coloured is equivalent to a check of whether the subgraph of $t(Z)$ induced by (vertices corresponding to) the Z-vertices of S can be 3-coloured. We know that our accepting computation on (G, v) outputs v. This can only happen if not all of $\{y, z, w\}$ are output, *i.e.*, if at least one a-vertex, a_m, say, is output, *i.e.*, if the Z-vertex m is not output, *i.e.*, if the graph $t(Z)$ can not be 3-coloured. The result follows. □

7.5 Conclusion

In this chapter we discussed the complexity of the problem MaxDegree($\mathcal{C}, \pi$) for properties π that are hereditary and non-trivial on $\mathcal{C}$. We considered properties testable in deterministic polynomial time, and obtained a class of **NP**-complete problems. We discussed the complexity of the problem MaxDegree(undirected graphs, 3-colourable), which we proved complete for $\mathbf{\Sigma_2^p}$. There are natural directions in which to further extend the research.

Modifying our algorithm so that it always chooses vertices of smallest degree we can similarly define the problem MinDegree(π). Does the complexity of MinDegree(π) mirror that of MaxDegree(π)? Note that our proof technique does not work with this problem.

Can we obtain a dichotomy result for the class of problems MaxDegree(undirected graphs, H-colourable), where H is an undirected graph?

These questions conclude our study of the complexity of problems related to greedy algorithms on ordered graphs. We think that there is scope for a considerable amount of further research in this field, and we will therefore continue its development in the future.

Bibliography

[1] S. S. Anderson, *Graph theory and finite combinatorics*, Markham Publishing Company (1970).

[2] T. Asano and T. Hirata, Edge-deletion and edge-contraction problems, *Proceedings of* 14*th Annual ACM Symposium on the Theory of Computing* (1982) 245–254.

[3] L.W. Beineke, On derived graphs and digraphs, in *Beitrage zur Graphen-Theorie* (H. Sachs, H. J. Voss and H. Walther, eds), Teubner, Berlin (1968) 17-23.

[4] C. Berge, *Graphs and Hypergraphs*, North-Holland (1973).

[5] N. L. Biggs, E. K. Lloyd and R. J. Wilson, *Graph Theory* 1736-1936, Clarendon Press (1976).

[6] D. Bovet and P. Crescenzi, *Introduction to the Theory of Complexity*, Prentice Hall (1994).

[7] G. Brassard and P. Bratley, *Fundamentals of Algorithmics*, Prentice Hall (1996).

[8] P. E. Ceruzzi, *A History of Modern Computing*, The MIT Press (1998).

[9] T. H. Cormen, C. E. Leiserson and R. L. Rivest, *Introduction to Algorithms*, McGraw-Hill (1991).

[10] J. Diaz, M. Serna and D. M. Thilikos, (H,C,K)-colorings: fast, easy and hard cases, *Proceedings 26th International Symposium on Mathematical Foundations of Computer Science* (MFCS-2001), Lecture Notes in Computer Science Vol. 2136, Springer-Verlag, Berlin (2001) 304-315.

[11] M. Dyer and C. Greenhill, The complexity of counting graph homomorphisms, *Random Structures and Algorithms* **17** (2000) 260–289.

[12] L. Euler, Solutio problematis ad geometriam situs pertinentis, *Commetarii Academiae Scientiarum Imperialis Petropolitanae* **8** (1736) 128–140.

[13] T. Feder and P. Hell, List homomorphisms to reflexive graphs, *Journal of Combinatorial Theory Series B* **72** (1998) 236–250.

[14] T. Feder, P. Hell and J. Huang, List homomorphisms and circular arc graphs, *Combinatorica* **19** (1999) 487-505.

[15] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman (1979).

[16] A. M. Gibbons and W. Rytter, *Efficient Parallel Algorithms*, Cambridge University Press (1988).

[17] R. Greenlaw, Ordered vertex removal and subgraph problems, *Journal of Computer and System Sciences* **39** (1989) 323–342.

[18] R. Greenlaw and H. J. Hoover, *Fundamentals of the Theory of Computation Principles and Practice*, Morgan Kaufmann (1998).

[19] R. Greenlaw, H. J. Hoover and W. L. Ruzzo, *Limits to Parallel Computation: P-Completeness Theory*, Oxford University Press (1995).

[20] J. Gross and J. Yellen, *Graph Theory and its Applications*, CRC Press (1999).

[21] F. Harary, *Graph Theory*, Addison-Wesley (1969).

[22] D. Harel, *Algorithmics : the Spirit of Computing*, Addison-Wesley (1987).

[23] P. Hell and J. Nesetril, On the complexity of H-colouring, *Journal of Combinatorial Theory Series B* **48** (1990) 92–110.

[24] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms*, Computer Science Press (1978).

[25] D. S. Johnson, M. Yannakakis and C. H. Papadimitriou, On generating all maximal independent sets, *Information Processing Letters* **27** (1988) 119–123.

[26] C. G. Lekkerkerker and J. C. Boland, Representation of a finite graph by a set of intervals on the real line, *Fundamenta Mathematicae* **51** (1962/1963) 45-64.

[27] J. M. Lewis and M. Yannakakis, The node deletion problem for hereditary properties is NP-complete, *Journal of Computer and System Sciences* **20** (1980) 219–230.

[28] U. Manber, *Introduction to Algorithms : A Creative Approach*, Addison-Wesley (1989).

[29] S. Miyano, The lexicographically first maximal subgraph problems: P-completeness and NC algorithms, *Mathematical Systems Theory* **22** (1989) 47–73.

[30] S. Miyano, $\mathbf{\Delta_2^p}$-complete lexicographically first maximal subgraph problems, *Theoretical Computer Science* **88** (1991) 33–57.

[31] C. H. Papadimitriou, *Computational Complexity*, Addison-Wesley, (1994).

[32] C. H. Papadimitriou, On the complexity of unique solutions, *Journal of the Association for Computing Machinery* **31** (1984) 392-400.

[33] A. Puricella and I. A. Stewart, A generic greedy algorithm, partially-ordered graphs and NP-completeness, *Proceedings of 27th International Workshop on Graph-Theoretic Concepts in Computer Science* (WG'01), Lecture Notes in Computer Science Vol. 2204, Springer-Verlag, Berlin (2001) 306-316.

[34] A. Puricella and I. A. Stewart, Greedy algorithms, H-colourings and a complexity-theoretic dichotomy, *Theoretical Computer Science*, to appear.

[35] F. P. Ramsey, On a problem of formal logic, *Proceeding of the London Mathematical Society,* **30** (1930) 264-286.

[36] I. A. Stewart, Complete problems involving Boolean labelled structures and projection translations, *Journal of Logic and Computation* **1** (1991) 861–882.

[37] L. J. Stockmeyer, The polynomial-time hierarchy, *Theoretical Computer Science* **3** (1977) 1-22.

[38] C. R. Subramanian and C. E. Veni Madhavan, The existence of homeomorphic subgraphs in chordal graphs, *Applied Mathematics Letters* **10** (1997) 17-22.

[39] J. van Leeuwen, *Handbook of Theoretical Computer Science A : Algorithms and Complexity*, Elsevier (1990).

[40] T. Watanabe, T. Ae and A. Nakamura, On the removal of forbidden graphs by edge-deletion or by edge-contraction, *Discrete Applied Mathematics* **3** (1981) 151–153.

[41] M. R. Williams, *A History of Computing Technology*, IEEE Computer Society Press (1997).

[42] M. Yannakakis, Node-deletion problems on bipartite graphs, *SIAM Journal of Computing* **10** (1981) 310–327.